US011693632B2

(12) United States Patent
Latapie et al.

(10) Patent No.: US 11,693,632 B2
(45) Date of Patent: Jul. 4, 2023

(54) **COLLABORATIVE VISUAL PROGRAMMING ENVIRONMENT WITH CUMULATIVE LEARNING USING A DEEP FUSION REASONING ENGINE**

(71) Applicant: **Cisco Technology, Inc.**, San Jose, CA (US)

(72) Inventors: **Hugo Latapie**, Long Beach, CA (US); **Ozkan Kilic**, Long Beach, CA (US); **Ramana Rao V. R. Kompella**, Cupertino, CA (US); **Pallavi Kalapatapu**, San Jose, CA (US); **Gaowen Liu**, Austin, TX (US)

(73) Assignee: **Cisco Technology, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 14 days.

(21) Appl. No.: **17/157,027**

(22) Filed: **Jan. 25, 2021**

(65) **Prior Publication Data**

US 2022/0236966 A1 Jul. 28, 2022

(51) **Int. Cl.**

G06F 8/34	(2018.01)
G06N 20/00	(2019.01)
G06N 5/04	(2023.01)

(52) **U.S. Cl.**

CPC ***G06F 8/34*** (2013.01); ***G06N 5/04*** (2013.01); ***G06N 20/00*** (2019.01)

(58) **Field of Classification Search**

CPC G06F 8/34; G06N 20/00; G06N 5/04

USPC 717/109

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

10,565,802	B2	2/2020	Grossmann et al.	
10,740,804	B2	8/2020	Spivack et al.	
10,887,197	B2	1/2021	Fenoglio et al.	
10,965,516	B2	3/2021	Fenoglio et al.	
2009/0031281	A1*	1/2009	Zhang	G06F 8/34 717/105
2016/0359687	A1*	12/2016	Gao	G06F 3/04817
2016/0378861	A1	12/2016	Eledath et al.	
2017/0109933	A1*	4/2017	Voorhees	G06F 11/36
2020/0285449	A1*	9/2020	McIntosh	G06N 3/0454
2020/0319856	A1*	10/2020	Loria	G06F 3/017
2021/0042532	A1	2/2021	Latapie et al.	

(Continued)

OTHER PUBLICATIONS

Valsamakis et al "Collaborative Visual Programming Workspace for Blockly", 2020, [Online], pp. 1-6, [Retrieved from internet on Feb. 1, 2023], <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9127253> (Year: 2020).*

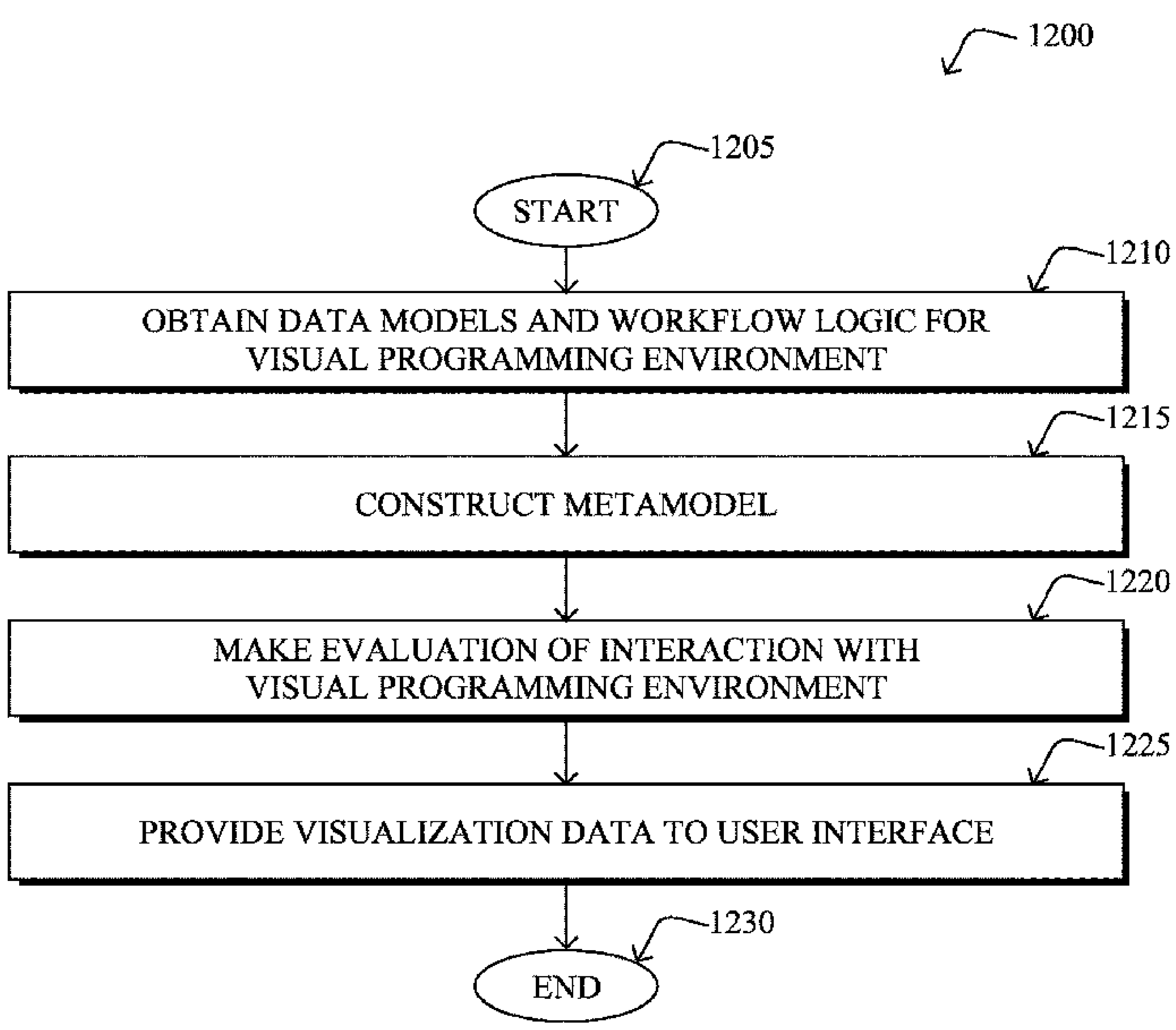
(Continued)

Primary Examiner — Anibal Rivera
Assistant Examiner — Zengpu Wei
(74) *Attorney, Agent, or Firm* — Behmke Innovation Group LLC; Kenneth J. Heywood; Jonathon P. Western

(57) **ABSTRACT**

In one embodiment, a device obtains data models and workflow logic for a visual programming environment. The device constructs, based on the data models and workflow logic for the visual programming environment, a metamodel that comprises a knowledge graph. The device makes, using the metamodel, an evaluation of an interaction between a user and the visual programming environment. The device provides, based on the evaluation, visualization data to a user interface of the visual programming environment.

18 Claims, 13 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

2021/0081841 A1*	3/2021	Sikka		G06N 3/048
2021/0097052 A1*	4/2021	Hans		G06N 5/003
2021/0174155 A1	6/2021	Smith et al.		
2021/0279615 A1	9/2021	Latapie et al.		
2021/0390423 A1	12/2021	Latapie et al.		
2022/0027900 A1*	1/2022	Suh		G06Q 20/4016

OTHER PUBLICATIONS

Agrawal, et al., "VQA: Visual Question Answering", Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2015, 25 pages, arXiv:1505.00468v7 [cs.CL].
Aleksander, Igor, "Machine consciousness" In Scholarpedia. 3(2):4162, Oct. 21, 2011, 7 pages.
Anderson, et al., "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering", 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Jun. 2018, pp. 6077-6086, IEEE, Salt Lake City, Utah.
Baudrillard, Jean, "Simulacra and Simulation", 1981, 159 pages, Galilee.
Baz, et al., "Context-aware hybrid classification system for fine-grained retail product recognition", 2016 IEEE 12th Image, Video, and Multidimensional Signal Processing Workshop (IVMSP), Jul. 2016, 5 pages, IEEE, Bordeaux, France.
Bělohlávek, Radim, "Concept lattices and order in fuzzy logic", Annals of Pure and Applied Logic 128 (2004) 277-298, Elsevier.
Box, G. E. P., "Science and Statistics", In Journal of the American Statistical Association, 71(356), Dec. 1976, pp. 791-799.
Chalmers, David J., "The Conscious Mind: In Search of a Fundamental Theory", 1996, 433 pages, Oxford University Press, New York.
Chella, et al., "A cognitive framework for imitation learning", Robotics and Autonomous Systems 54, Mar. 2006, pp. 403-408, Elsevier.
Chella, et al., "Artificial Consciousness", Chapter 20, In Perception-Action Cycle, 2011, pp. 637-671, Springer, New York.
Chella, et al., "Machine Consciousness: A Manifesto for Robotics", In International Journal of Machine Consciousness, 1(1), Jun. 2009, pp. 33-51, World Scientific Publishing Company.
Cohen, Paul R., "Projections as Concepts", Computer Science Department Faculty Publication Series (194), https://scholarworks.umass.edu/cs/_faculty/_pubs/194, 1997, 6 pages, University of Massachusetts, Amherst.
Cui, et al., "A survey on network embedding", IEEE Transactions on Knowledge and Data Engineering, vol. 31, Issue: 5, May 1, 2019, pp. 833-852, IEEE.
De Bono, Edward, "The Mechanism of Mind", 1967, 276 pages, Penguin Books.
Düntsch, et al., "Modal-style operators in qualitative data analysis", 2002 IEEE International Conference on Data Mining, 2002. Proceedings, Dec. 2002, pp. 155-162, IEEE, Maebashi City, Japan.
Franco, et al., "Grocery product detection and recognition", Expert Systems With Applications 81 (2017), pp. 163-176, Elsevier Ltd.
Gärdenfors, Peter, "Conceptual Spaces: The Geometry of Thought", 2000, 398 pages, MIT Press.
George, et al., "Recognizing Products: A Per-exemplar Multi-label Image Classification Approach", ECCV 2014, Part II, LNCS 8690, 2014, pp. 440-455, Springer International Publishing Switzerland.
Goertzel, et al., "CogPrime Architecture for Embodied Artificial General Intelligence", 2013 IEEE Symposium on Computational Intelligence for Human-like Intelligence (CIHLI), Apr. 2013, pp. 60-67, IEEE, Singapore.
Goertzel, Ben, "OpenCogPrime: A Cognitive Synergy Based Architecture for Artificial General Intelligence", 2009 8th IEEE International Conference on Cognitive Informatics, Jun. 2009, pp. 60-68, IEEE, Hong Kong, China.
Gorban, et al., "Blessing of dimensionality: mathematical foundations of the statistical physics of data", Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 376.2118, Jan. 2008, 18 pages, The Royal Society Publishing.
Grover, et al., "node2vec: Scalable Feature Learning for Networks", KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Aug. 2016, pp. 855-864, Association for Computing Machinery, New York, NY.
Hamilton, et al., "Representation Learning on Graphs: Methods and Applications", Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2017, 23 pages, IEEE.
Hammer, et al., "A Reasoning Based Model for Anomaly Detection in the Smart City Domain", IntelliSys 2020, AISC 1251, pp. 144-159, 2021, Springer Nature Switzerland AG.
Hobbs, Jerry R., "Granularity", In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, 1985, pp. 432-435, Morgan Kaufmann.
Horowitz, Alexandra, "Smelling themselves: Dogs investigate their own odours longer when modified in an "olfactory mirror" test", Behavioural Processes, 2017, 41 pages.
Johnson, Mark, "The Body in The Mind", 1987, 268 pages, The University of Chicago Press.
Kiryati, et al., "A probabilistic Hough transform", Pattern Recognition. 24(4), 1991, pp. 303-316, The Pattern Recognition Society.
Korzybski, Alfred, "Manhood of Humanity, The Science and Art of Human Engineering", 1921, 240 pages, E. P. Dutton & Company, New York, NY.
Korzybski, Alfred, "Science and Sanity: An Introduction to Non-Aristotelian Systems and General Semantics", 5th Edition, 1994, 910 pages, Institute of General Semantics, New York, NY.
Lakoff, G., "Women, Fire, and Dangerous Things", 1984, 631 pages, University of Chicago Press.
Li, et al., "Concept learning via granular computing: A cognitive viewpoint", Information Sciences 298 (2015), Published Dec. 2014, pp. 447-467, Elsevier Inc.
Lieto, et al., "Conceptual Spaces for Cognitive Architectures: A Lingua Franca for Different Levels of Representation", Biologically Inspired Cognitive Architectures 19, May 2017, 17 pages, Cognitive Robotics and Social Sensing Lab.
Ma, et al., "Granular computing and Dual Galois Connection", Information Sciences, 177(23), 2007, pp. 5365-5377, Elsevier Inc.
Macaulay, Thomas, "Facebook's chief AI scientist says GPT-3 is 'not a very good' Q&A system", online: https://thenextweb.com/news/facebooks-yann-lecun-says-gpt-3-is-not-very-good-as-a-qa-or-dialog-system, Oct. 28, 2020, accessed Nov. 18, 2021, 3 pages.
Murahari, et la., "Improving Generative Visual Dialog by Answering Diverse Questions", Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, Nov. 2018, pp. 1449-1454, Association for Computational Linguistics, Hong Kong, China.
Patel, et al., "Video Representation and Suspicious Event Detection Using Semantic Technologies", online: http://semantic-web-journal.net/system/files/swj2427.pdf, Semantic Web 0, Sep. 10, 2020, accessed Aug. 9, 2021, 25 pages, IOS Press.
Pauli, Wolfgang, "Part I. General: (A) theory. Some relations between electrochemical behaviour and the structure of colloids", Jan. 1935, pp. 11-27, Transactions of the Faraday Society, vol. 1.
Scarselli, et al., "The Graph Neural Network Model", IEEE Transactions on Neural Networks (vol. 20, Issue: 1, Jan. 2009), pp. 61-80, IEEE.
Searle, John R., "The Rediscovery of the Mind", 1992, 104 pages, MIT Press.
Speer, et al., "ConceptNet 5.5: An Open Multilingual Graph of General Knowledge", online: https://arxiv.org/pdf/1612.03975.pdf, 2017, 9 pagers, Association for the Advancement of Artificial Intelligence.
Tan, et al., "EfficientDet: Scalable and Efficient Object Detection", online: https://arxiv.org/pdf/1911.09070.pdf, Jul. 2020, 10 pages.
Taylor, J. G., "CODAM: A neural network model of consciousness", Neural Networks 20 (2007), pp. 983-992, Elsevier Ltd.
Taylor, J. G., "William James on Consciousness Beyond the Margin", 1996, 231 pages, Princeton University Press.

(56) **References Cited**

OTHER PUBLICATIONS

Thórisson, et al., "Cumulative Learning", Artificial General Intelligence—12th International Conference, AGI 2019, Proceedings, pp. 198-208, Springer.
Thórisson, Kristinn R., "A New Constructivist AI: From Manual Methods to Self-Constructive Systems", Chapter 9, Apr. 2012, pp. 147-174, Atlantis Press Book.
Thórisson, Kristinn R., "Integrated AI Systems", Minds & Machines 17, Mar. 2007, pp. 11-25.
Tonioni, et al., "Product recognition in store shelves as a sub-graph isomorphism problem", online: https://arxiv.org/abs/1707.08378, Sep. 2017, 14 pages.
Unger, et al., "The Singular Universe and the Reality of Time: A Proposal in Natural Philosophy", 2015, 558 pages, Cambridge University Press.
Unger, R. M. 2014. "Roberto Unger: Free Classical Social Theory from Illusions of False Necessity", Online Lecture. 45 pages Retrieved on Nov. 22, 2021 from https://www.youtube.com/watch?v=yYOOwNRFTcY.
Wang, et al., "Concept Analysis via Rough Set and AFS Algebra", Information Sciences 178 (2008), pp. 4125-4137, Elsevier Inc.
Wang, Pei, "Experience-grounded semantics: a theory for intelligent systems", Aug. 2004, 33 pages, Elsevier Science.
Wang, Pei, "Insufficient Knowledge and Resources—A Biological Constraint and Its Functional Implications", Biologically Inspired Cognitive Architectures II: Papers from the AAAI Fall Symposium (FS-09-01), 2009, pp. 188-193, Association for the Advancement of Artificial Intelligence (www.aaai.org).
Wang, Pei, "Non-axiomatic logic (nal) specification", online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.167.2069&rep=rep1&type=pdf, Oct. 2009, 88 pages.
Wang, Pei, "On Defining Artificial Intelligence", Journal of Artificial General Intelligence 10(2) 2019, pp. 1-37, Sciendo.
Wang, Pei, "Rigid Flexibility—The Logic of Intelligence", Draft for Comment, Feb. 8, 2004, 329 pages, mindspring.com.
Wang, et al. "Self in NARS, an AGI System", vol. 5, Article 20, Mar. 2018, 15 pages, Frontiers in Robotics and AI.
Wang, et al., "SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems", online: https://arxiv.org/pdf/1905.00537.pdf, 2019, 29 pages, 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.
Wille, Rudolf, "Restructuring Lattice Theory: An Approach Based on Heirarchies of Concepts", I. Rival (Ed.), Ordered Sets, 1982, pp. 314-339.
Yao, et al., "A Granular Computing Paradigm for Concept Learning", Emerging Paradigms in Machine Learning, Springer, London, pp. 307-326, 2012.
Yao, Y. Y., "Information Granulation and Rough Set Approximation", International Journal of Intelligent Systems, vol. 16, No. 1, 87-104, 2001.
Yao, Y. Y., "Integrative levels of granularity", Human-Centric Information Processing Through Granular Modelling, 2009, 20 pages, Studies in Computational Intelligence, vol. 182. Springer, Berlin, Heidelberg.
Ying, et al., "Graph convolutional neural networks for web-scale recommender systems", online: https://arxiv.org/pdf/1806.01973.pdf, In KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Aug. 19-23, 2018, London, United Kingdom. ACM, New York, NY, USA, 10 pages.
Zhou, et al., "Graph neural networks: A review of methods and applications", AI Open, 2020, pp. 57-81, Elsevier B.V.
Zhu, et al., "Describing Unseen Videos via Multi-modal Cooperative Dialog Agents" Computer Vision—ECCV 2020, 17 pages, Lecture Notes in Computer Science, vol. 12368. Springer.
Rokhsaritalemi, et al., "A Review on Mixed Reality: Current Trends, Challenges and Prospects", 10(2):636, Jan. 2020, 26 pages, Applied Sciences.
Lampropoulos, et al., "Enhancing the Functionality of Augmented Reality Using Deep Learning, Semantic Web and Knowledge Graphs: A Review", Visual Informatics 4, Jan. 2020, pp. 32-42, Elsevier B.V.
Myers, Brad A., "Taxonomies of visual programming and program visualization", Journal of Visual Languages & Computing, vol. 1, Issue 1, Mar. 1990, pp. 97-123, Elsevier.
Green, et al., "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework", Journal of Visual Languages & Computing, vol. 7, Issue 2, Jun. 1996, pp. 131-174, Elsevier.
Latapie, et al., "A Metamodel and Framework for AGI", online: https://arxiv.org/abs/2008.12879, Sep. 2020, 14 pages, arXiv.org.
Atzeni, et al. , "Translating Natural Language to Code: An Unsupervised Ontology-Based Approach", 2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), Sep. 2018, 8 pages, IEEE.
"Haskell (Programming Language)", online: https://en.wikipedia.org/wiki/Haskell_(programming_language), 9 pages, Dec. 2020, Wikimedia Foundation, Inc.

* cited by examiner

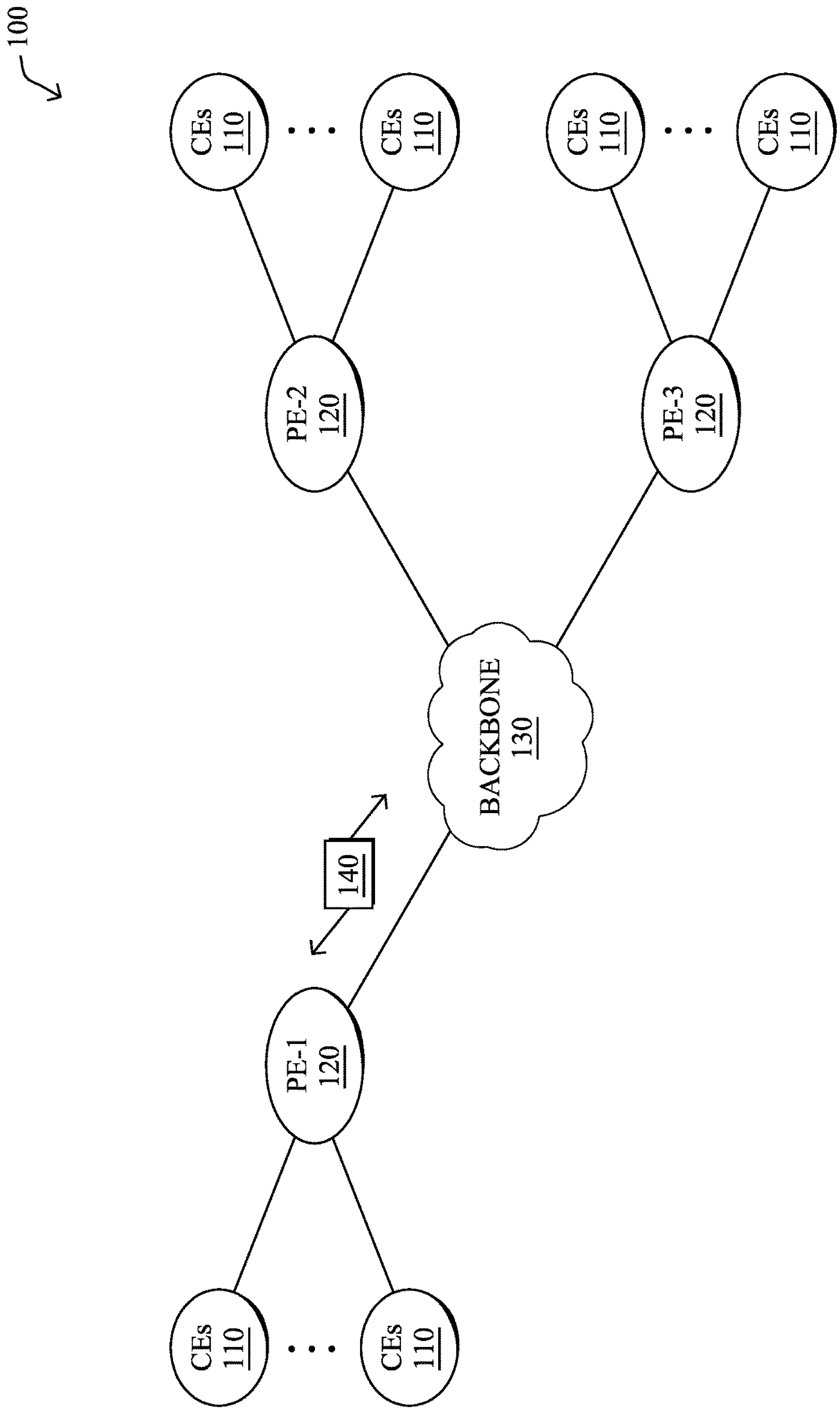

100
CEs 110
CEs 110
CEs 110
CEs 110
PE-2 120
PE-3 120
BACKBONE 130
140
PE-1 120
CEs 110
CEs 110

FIG. 1A

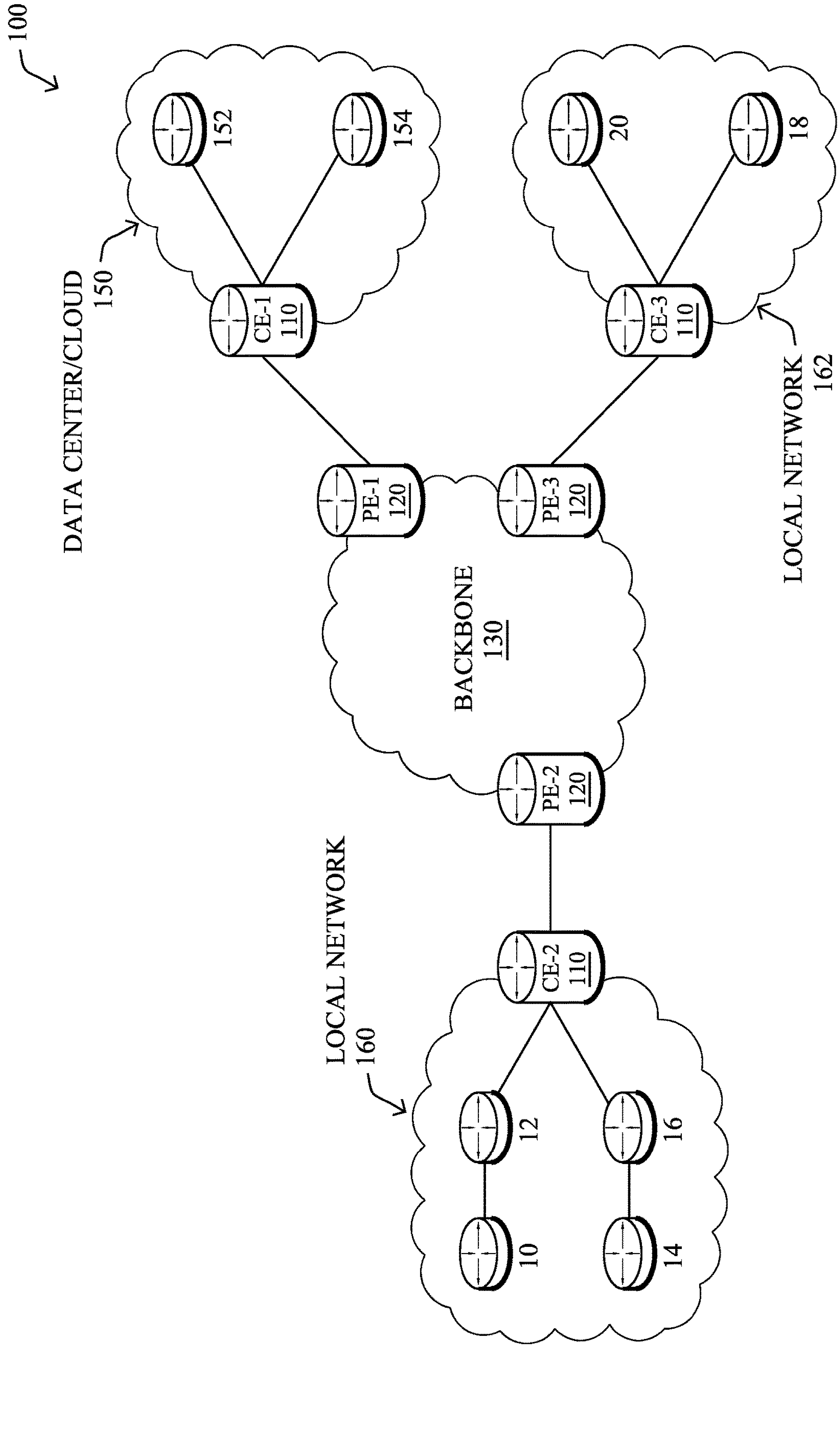

100
DATA CENTER/CLOUD
150
152
154
CE-1
110
PE-1
120
BACKBONE
130
PE-3
120
CE-3
110
20
18
LOCAL NETWORK
162
PE-2
120
CE-2
110
LOCAL NETWORK
160
12
16
10
14

FIG. 1B

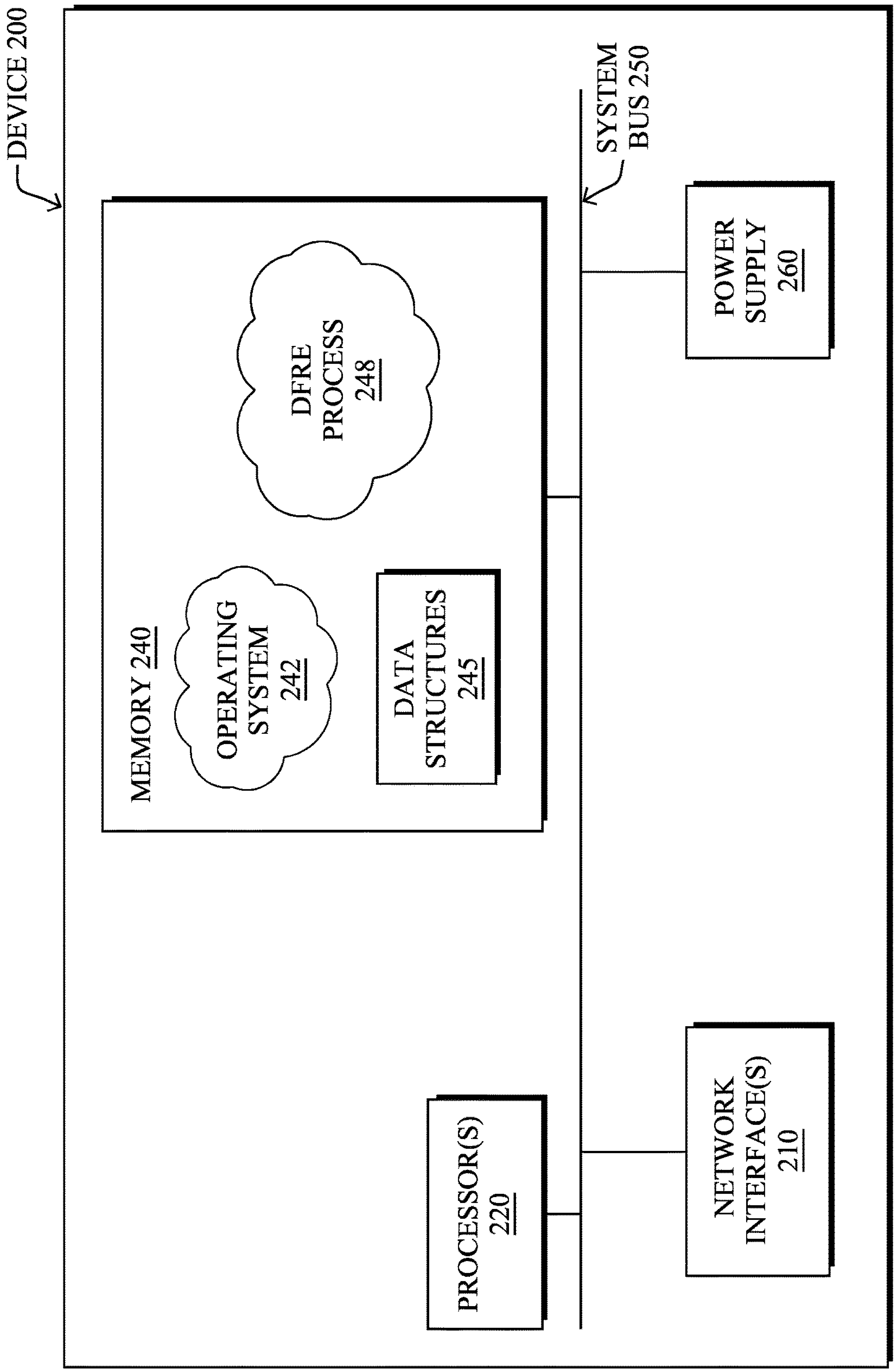

DEVICE 200
MEMORY 240
OPERATING SYSTEM 242
DATA STRUCTURES 245
DFRE PROCESS 248
SYSTEM BUS 250
POWER SUPPLY 260
PROCESSOR(S) 220
NETWORK INTERFACE(S) 210

FIG. 2

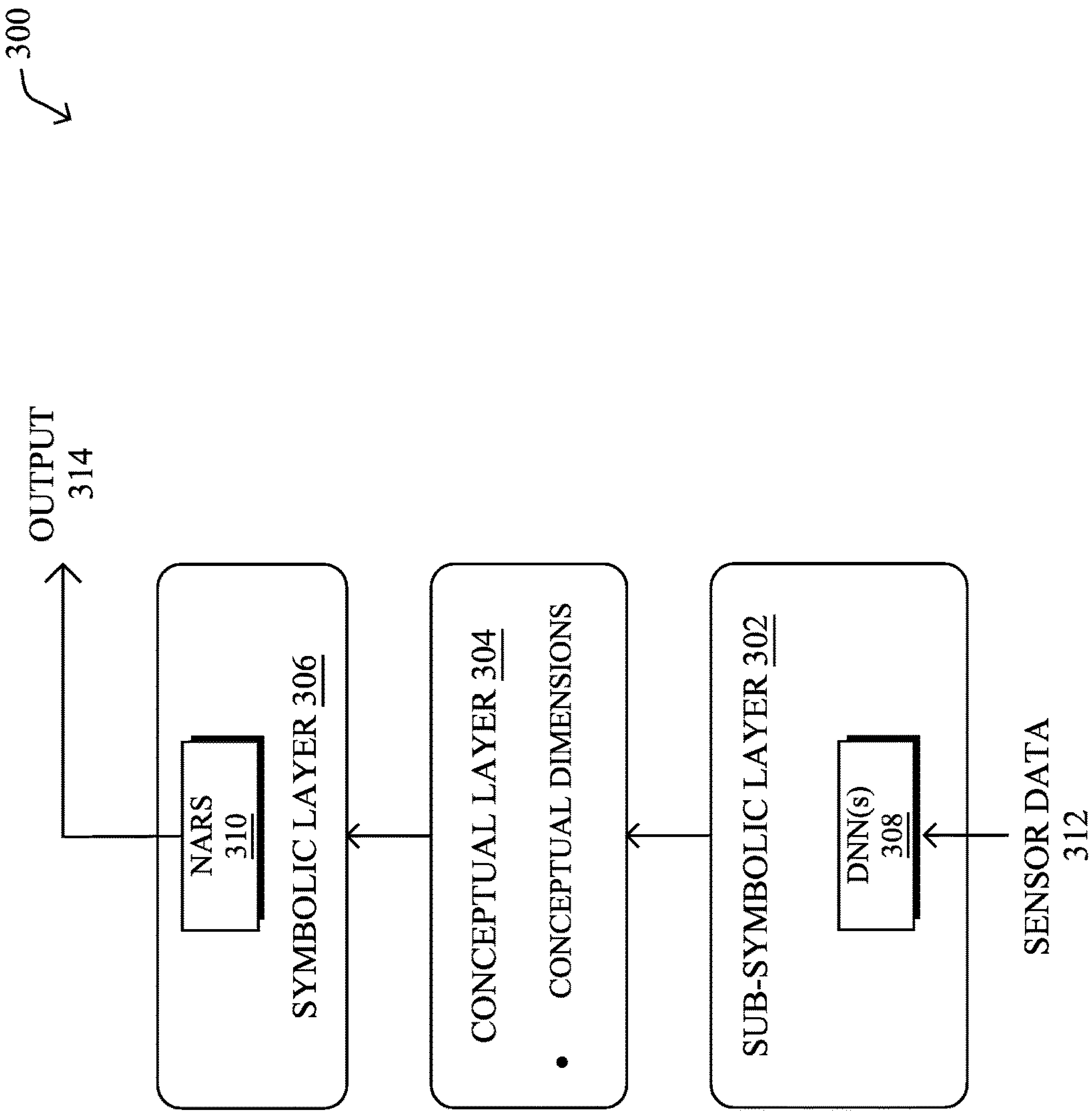

300
OUTPUT
314
NARS
310
SYMBOLIC LAYER 306
CONCEPTUAL LAYER 304
• CONCEPTUAL DIMENSIONS
SUB-SYMBOLIC LAYER 302
DNN(s)
308
SENSOR DATA
312

FIG. 3

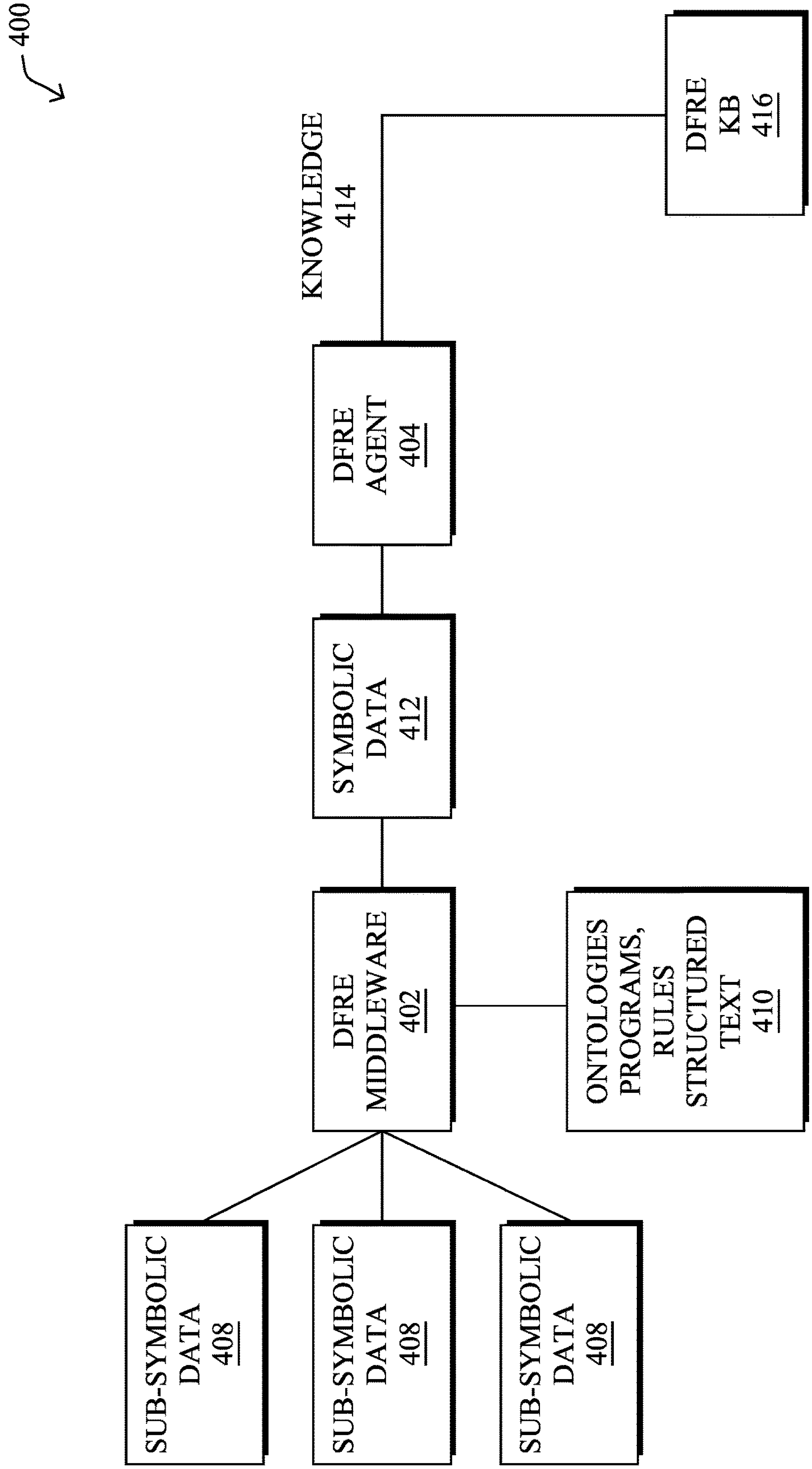

400
KNOWLEDGE
414
DFRE
KB
416
DFRE
AGENT
404
SYMBOLIC
DATA
412
DFRE
MIDDLEWARE
402
ONTOLOGIES
PROGRAMS,
RULES
STRUCTURED
TEXT
410
SUB-SYMBOLIC
DATA
408
SUB-SYMBOLIC
DATA
408
SUB-SYMBOLIC
DATA
408

FIG. 4

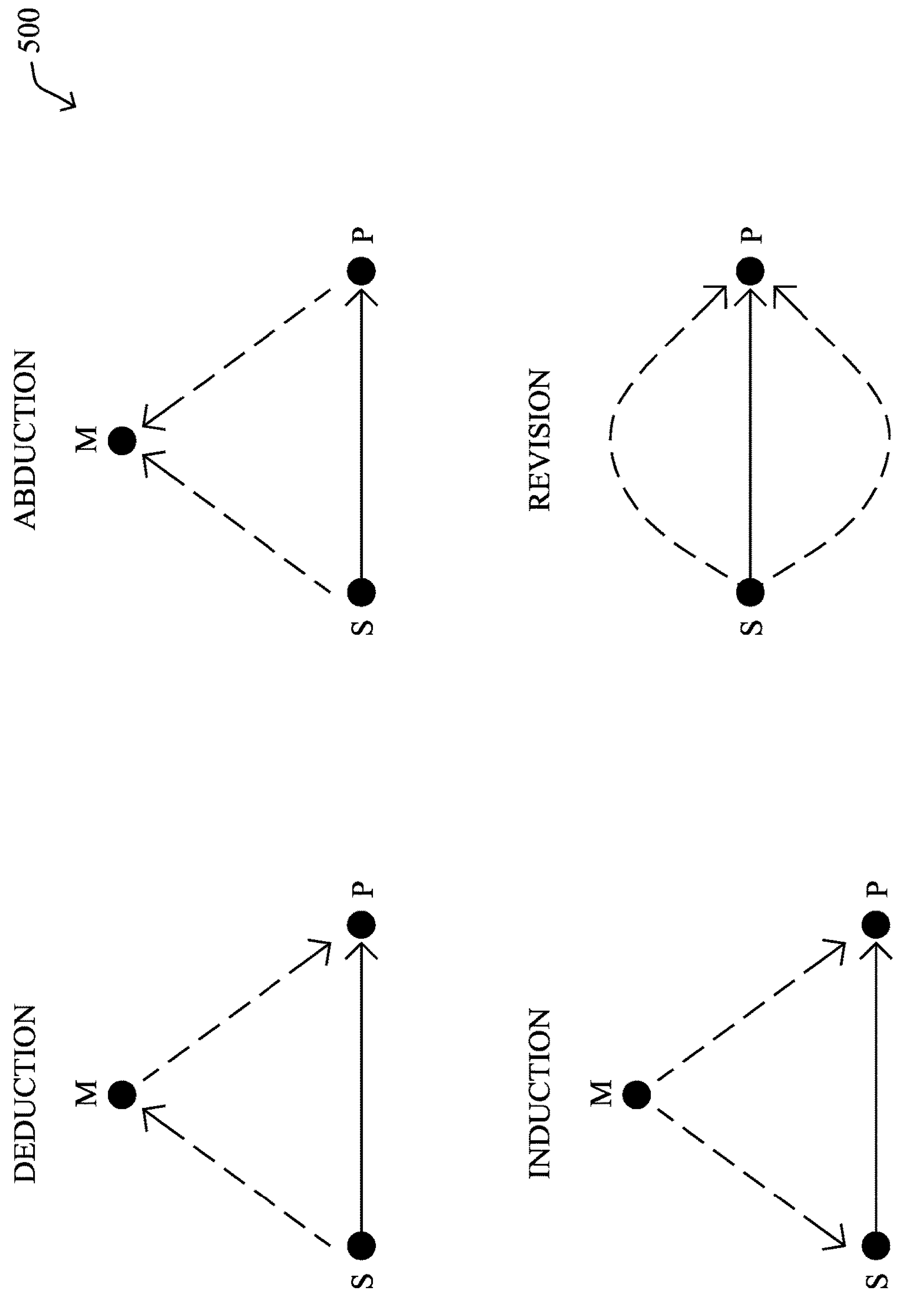


FIG. 5

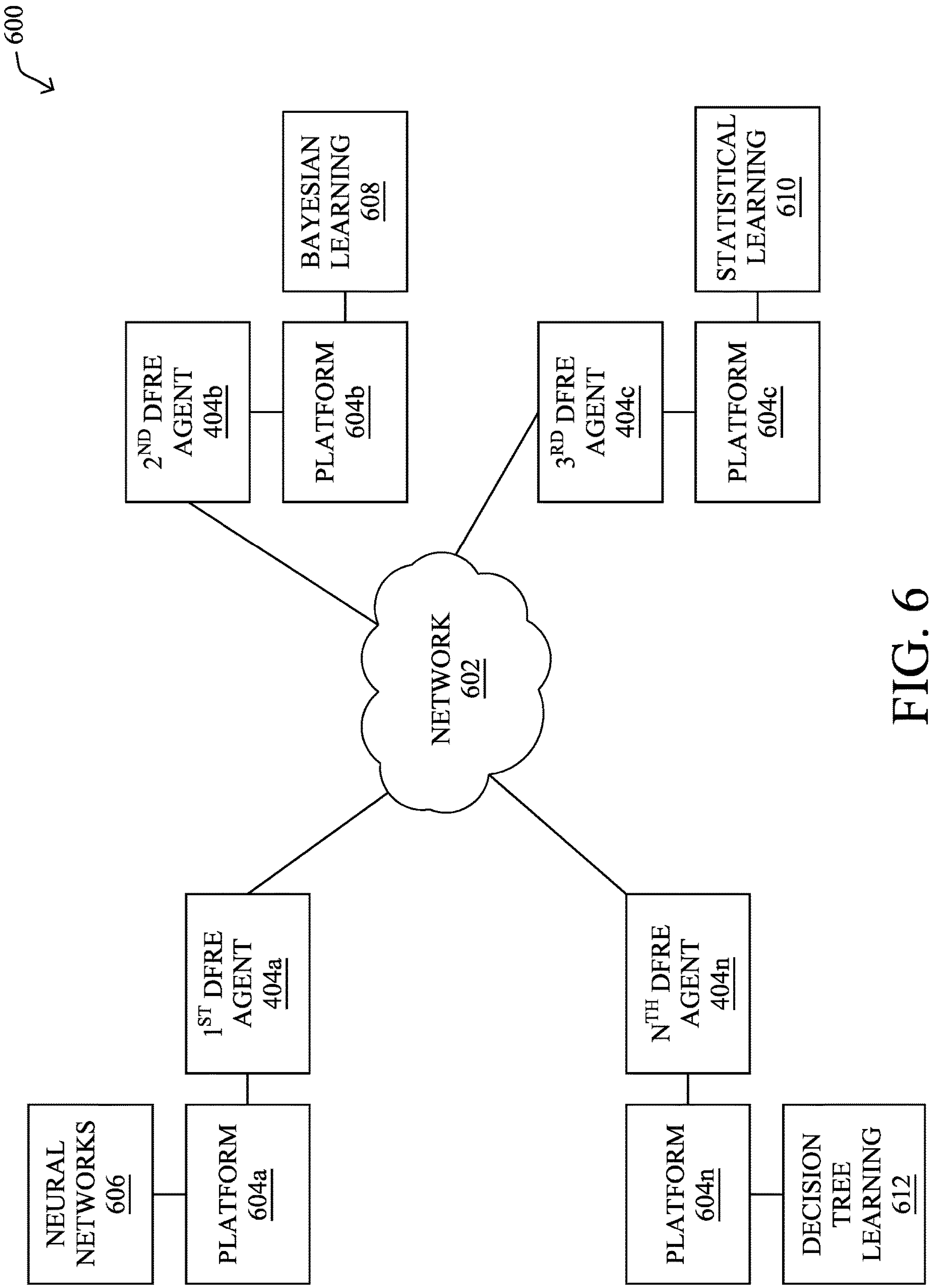

600
BAYESIAN LEARNING 608
STATISTICAL LEARNING 610
2ND DFRE AGENT 404b
PLATFORM 604b
3RD DFRE AGENT 404c
PLATFORM 604c
NETWORK 602
1ST DFRE AGENT 404a
NTH DFRE AGENT 404n
NEURAL NETWORKS 606
PLATFORM 604a
PLATFORM 604n
DECISION TREE LEARNING 612

FIG. 6

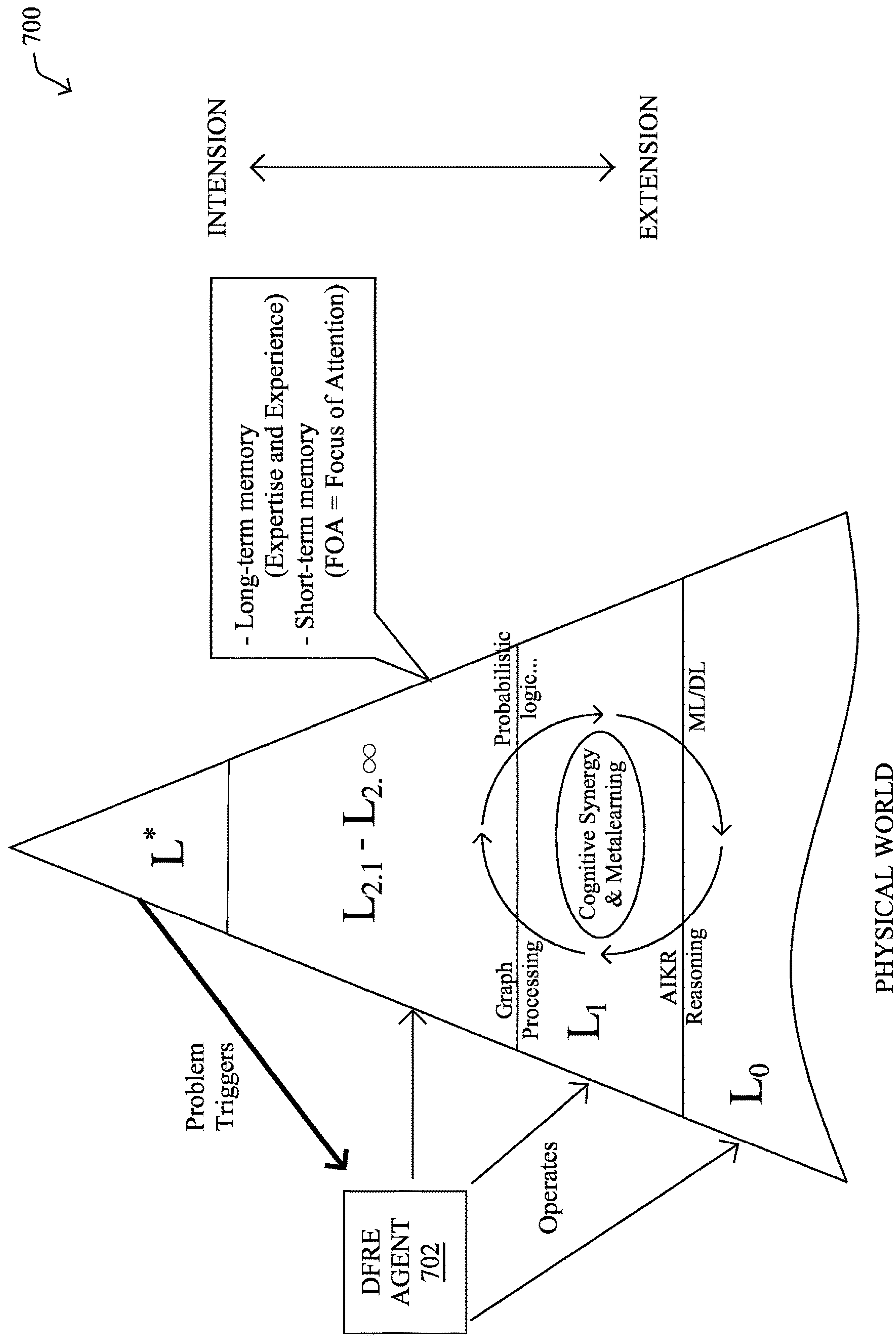

700
INTENSION
EXTENSION
- Long-term memory
(Expertise and Experience)
- Short-term memory
(FOA = Focus of Attention)
L^*
$L_{2.1} - L_{2.\infty}$
Probabilistic logic...
ML/DL
Cognitive Synergy & Metalearning
Graph Processing
AIKR Reasoning
L_1
L_0
PHYSICAL WORLD
Problem Triggers
Operates
DFRE AGENT 702

FIG. 7

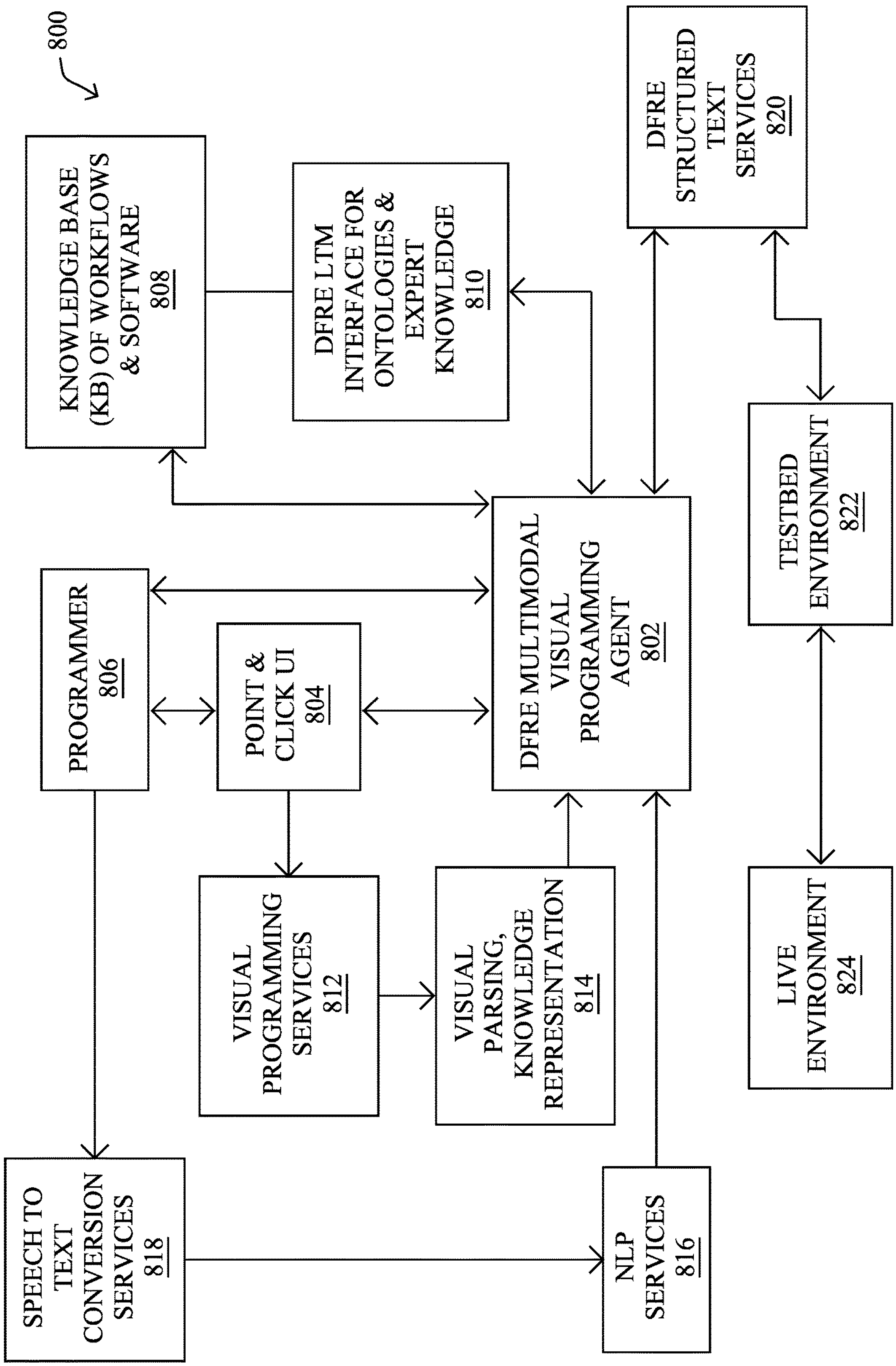

800
KNOWLEDGE BASE (KB) OF WORKFLOWS & SOFTWARE 808
DFRE LTM INTERFACE FOR ONTOLOGIES & EXPERT KNOWLEDGE 810
DFRE STRUCTURED TEXT SERVICES 820
PROGRAMMER 806
POINT & CLICK UI 804
DFRE MULTIMODAL VISUAL PROGRAMMING AGENT 802
TESTBED ENVIRONMENT 822
VISUAL PROGRAMMING SERVICES 812
VISUAL PARSING, KNOWLEDGE REPRESENTATION 814
LIVE ENVIRONMENT 824
SPEECH TO TEXT CONVERSION SERVICES 818
NLP SERVICES 816

FIG. 8

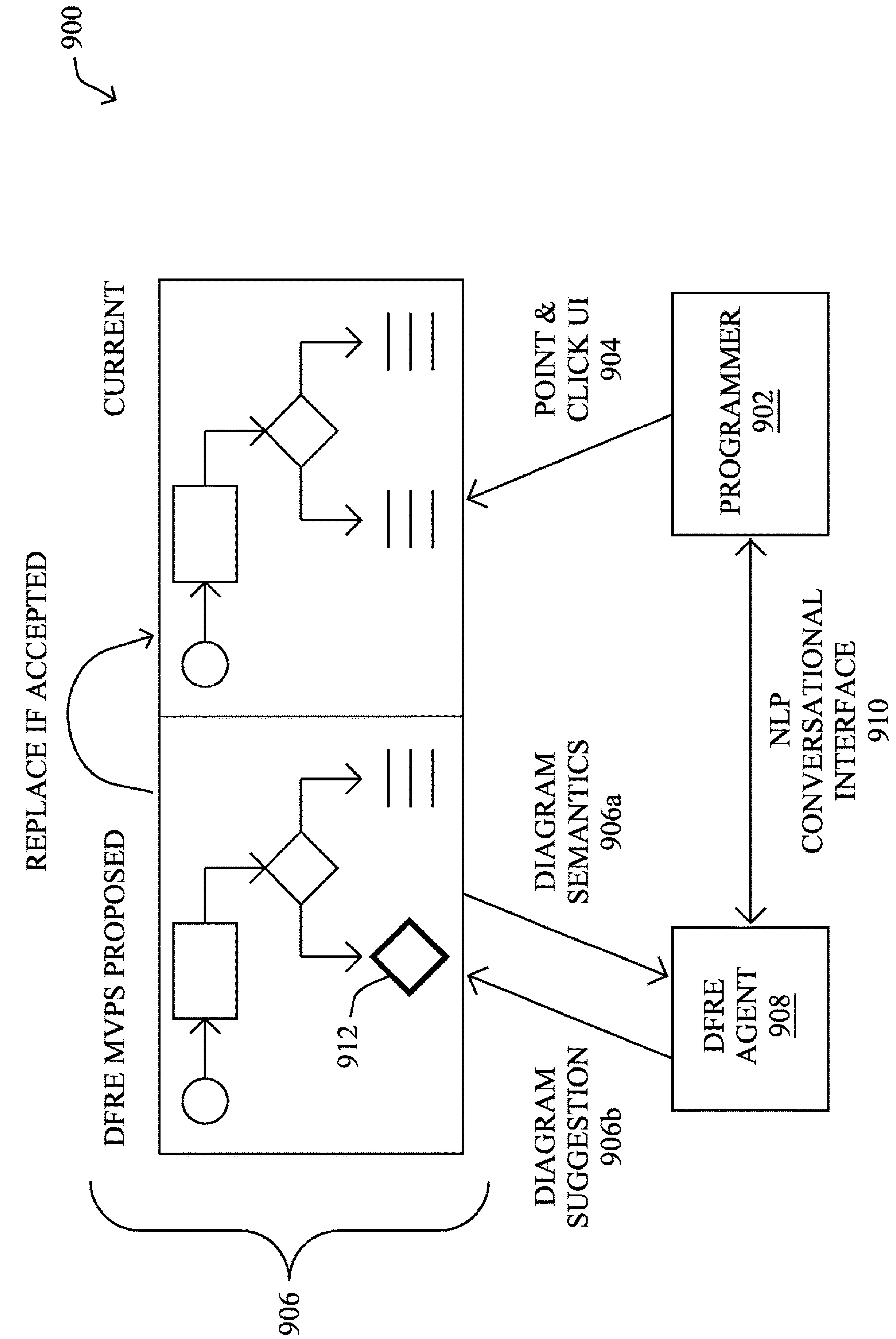

900
REPLACE IF ACCEPTED
DFRE MVPS PROPOSED
CURRENT
912
906
POINT & CLICK UI 904
PROGRAMMER 902
DIAGRAM SEMANTICS 906a
DIAGRAM SUGGESTION 906b
NLP CONVERSATIONAL INTERFACE 910
DFRE AGENT 908

FIG. 9

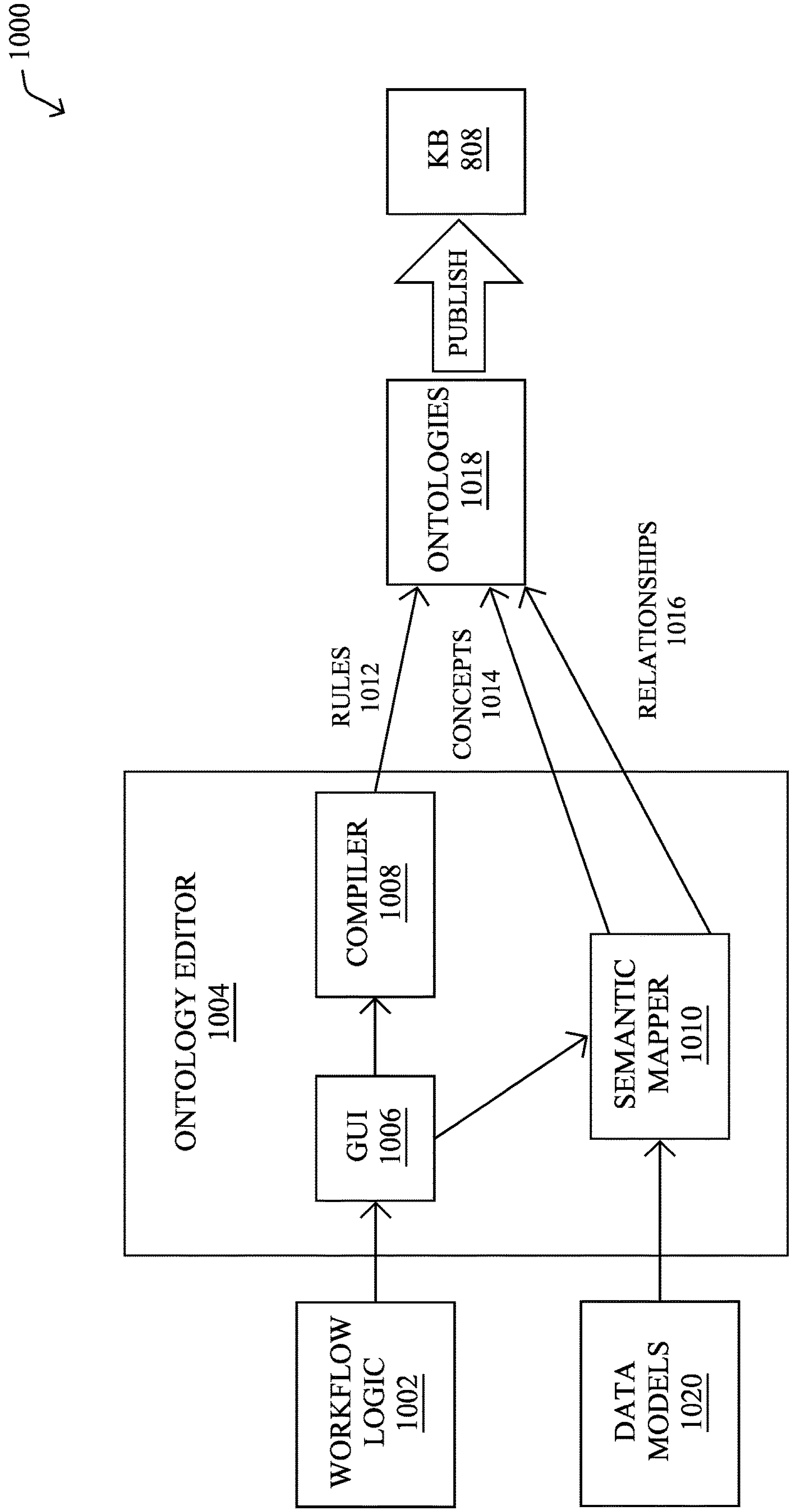

1000
KB
808
PUBLISH
ONTOLOGIES
1018
RULES
1012
CONCEPTS
1014
RELATIONSHIPS
1016
ONTOLOGY EDITOR
1004
COMPILER
1008
GUI
1006
SEMANTIC
MAPPER
1010
WORKFLOW
LOGIC
1002
DATA
MODELS
1020

FIG. 10

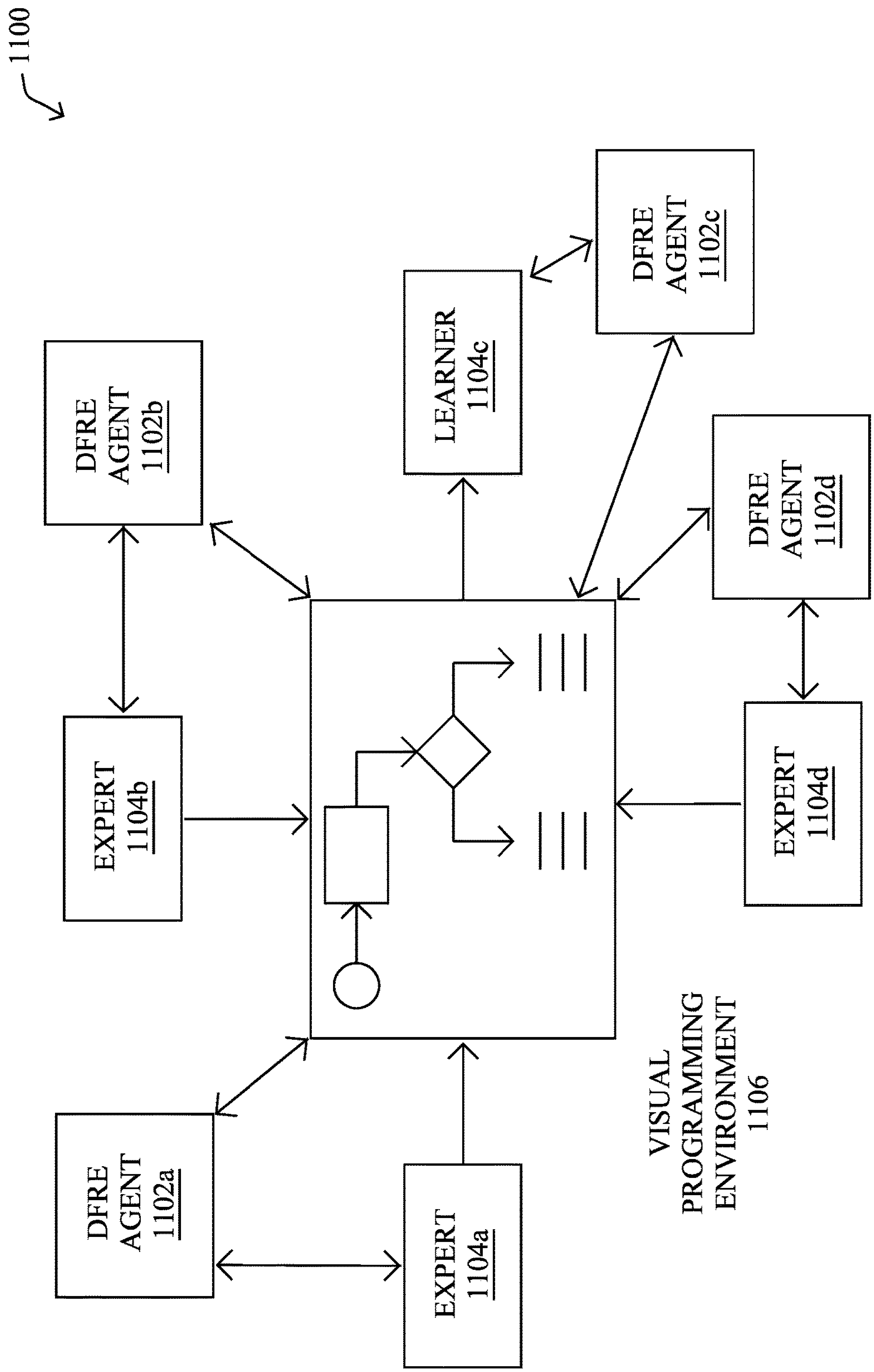

1100
DFRE AGENT 1102b
LEARNER 1104c
DFRE AGENT 1102c
DFRE AGENT 1102d
EXPERT 1104b
EXPERT 1104d
DFRE AGENT 1102a
EXPERT 1104a
VISUAL PROGRAMMING ENVIRONMENT 1106

FIG. 11

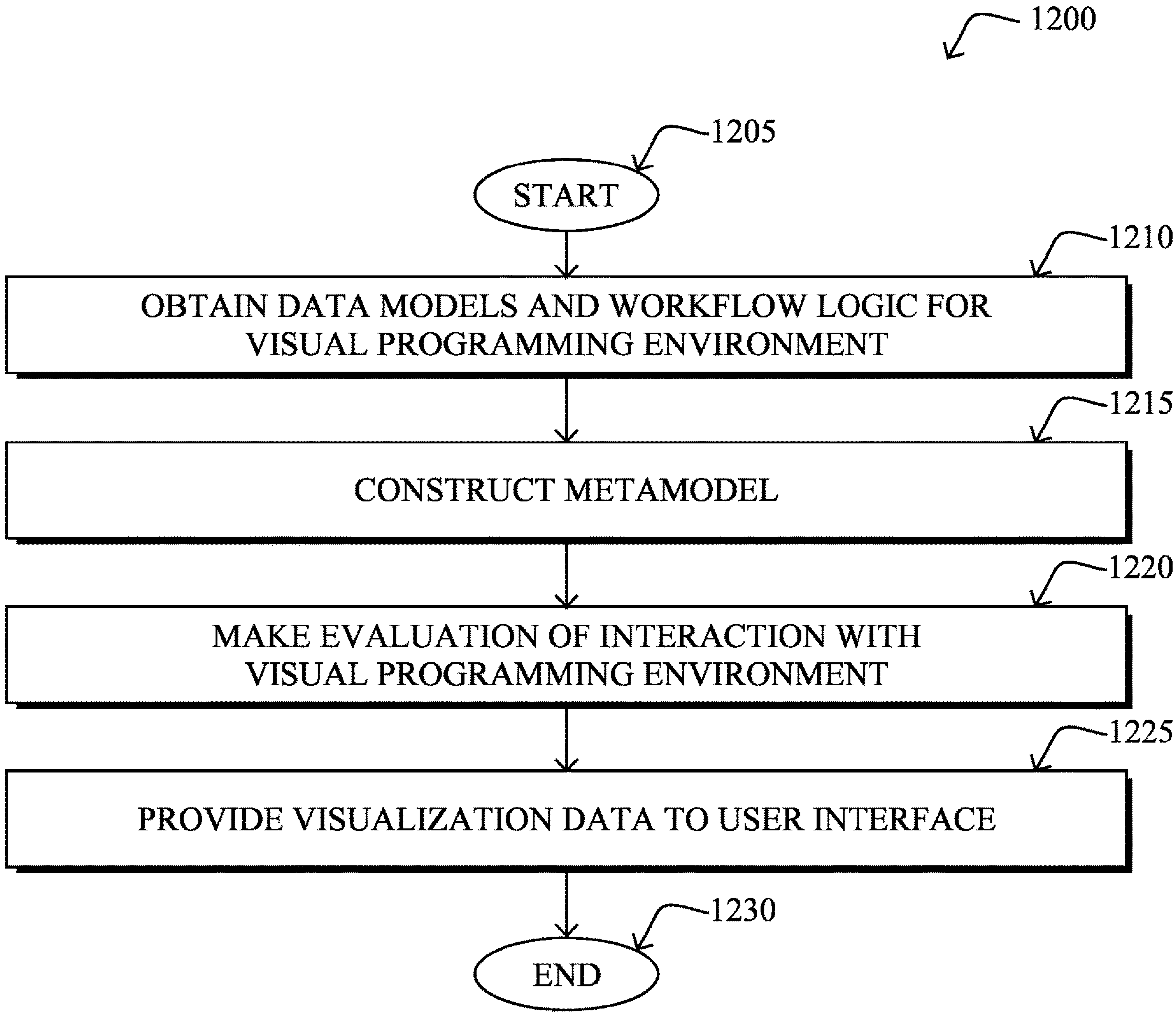

1200
1205
START
1210
OBTAIN DATA MODELS AND WORKFLOW LOGIC FOR VISUAL PROGRAMMING ENVIRONMENT
1215
CONSTRUCT METAMODEL
1220
MAKE EVALUATION OF INTERACTION WITH VISUAL PROGRAMMING ENVIRONMENT
1225
PROVIDE VISUALIZATION DATA TO USER INTERFACE
1230
END

FIG. 12

COLLABORATIVE VISUAL PROGRAMMING ENVIRONMENT WITH CUMULATIVE LEARNING USING A DEEP FUSION REASONING ENGINE

TECHNICAL FIELD

The present disclosure relates generally to computer networks, and, more particularly, to a collaborative visual programming environment with cumulative learning using a deep fusion reasoning engine.

BACKGROUND

The creation of models, ontologies, diagrams, software programs, and other similar artifacts remains a very time consuming and resource intensive activity. While there have been many attempts at creating visual programming systems, these have largely failed due to the relatively low productivity that results from their point-and-click interfaces. Another source of failure of these visual programming systems has been caused by the referential integrity problem. More specifically, once an artifact is created via the visual programming system, any editing of the artifact cannot be automatically parsed into the visual representation.

Although new computer languages with higher expressive power are continually being introduced, e.g., functional programming languages such as Haskell and Rust, these more expressive languages also require increasing sophistication on the part of the software developer. This limits the usability of these more expressive languages. Indeed, the gap between the supply of computer scientists, programmers, and other experts that can produce high quality digital artifacts and the demand for such individuals is growing rapidly.

BRIEF DESCRIPTION OF THE DRAWINGS

The embodiments herein may be better understood by referring to the following description in conjunction with the accompanying drawings in which like reference numerals indicate identically or functionally similar elements, of which:

FIGS. **1A-1B** illustrate an example computer network;

FIG. **2** illustrates an example network device/node;

FIG. **3** illustrates an example hierarchy for a deep fusion reasoning engine (DFRE);

FIG. **4** illustrates an example DFRE architecture;

FIG. **5** illustrates an example of various inference types;

FIG. **6** illustrates an example architecture for multiple DFRE agents;

FIG. **7** illustrates an example DFRE metamodel;

FIG. **8** illustrates an example DFRE-based architecture for a visual programming environment;

FIG. **9** illustrates an example of the interaction of a DFRE system with a visual programming environment;

FIG. **10** illustrates an example data pipeline to populate a DFRE knowledge base;

FIG. **11** illustrates an example of the collaboration of users in a visual programming environment; and

FIG. **12** illustrates an example simplified procedure for using a DFRE with a visual programming environment.

DESCRIPTION OF EXAMPLE EMBODIMENTS

Overview

According to one or more embodiments of the disclosure, a device obtains data models and workflow logic for a visual programming environment. The device constructs, based on the data models and workflow logic for the visual programming environment, a metamodel that comprises a knowledge graph. The device makes, using the metamodel, an evaluation of an interaction between a user and the visual programming environment. The device provides, based on the evaluation, visualization data to a user interface of the visual programming environment.

Description

A computer network is a geographically distributed collection of nodes interconnected by communication links and segments for transporting data between end nodes, such as personal computers, cellular phones, workstations, or other devices, such as sensors, etc. Many types of networks are available, with the types ranging from local area networks (LANs) to wide area networks (WANs). LANs typically connect the nodes over dedicated private communications links located in the same general physical location, such as a building or campus. WANs, on the other hand, typically connect geographically dispersed nodes over long-distance communications links, such as common carrier telephone lines, optical lightpaths, synchronous optical networks (SONET), or synchronous digital hierarchy (SDH) links, or Powerline Communications (PLC) such as IEEE 61334, IEEE P1901.2, and others. The Internet is an example of a WAN that connects disparate networks throughout the world, providing global communication between nodes on various networks. The nodes typically communicate over the network by exchanging discrete frames or packets of data according to predefined protocols, such as the Transmission Control Protocol/Internet Protocol (TCP/IP). In this context, a protocol consists of a set of rules defining how the nodes interact with each other. Computer networks may be further interconnected by an intermediate network node, such as a router, to forward data from one network to another.

Smart object networks, such as sensor networks, in particular, are a specific type of network having spatially distributed autonomous devices such as sensors, actuators, etc., that cooperatively monitor physical or environmental conditions at different locations, such as, e.g., energy/power consumption, resource consumption (e.g., water/gas/etc. for advanced metering infrastructure or "AMI" applications) temperature, pressure, vibration, sound, radiation, motion, pollutants, etc. Other types of smart objects include actuators, e.g., responsible for turning on/off an engine or perform other actions. Sensor networks, a type of smart object network, are typically shared-media networks, such as wireless or PLC networks. That is, in addition to one or more sensors, each sensor device (node) in a sensor network may generally be equipped with a radio transceiver or other communication port such as PLC, a microcontroller, and an energy source, such as a battery. Often, smart object networks are considered field area networks (FANs), neighborhood area networks (NANs), personal area networks (PANs), etc. Generally, size and cost constraints on smart object nodes (e.g., sensors) result in corresponding constraints on resources such as energy, memory, computational speed and bandwidth.

FIG. **1A** is a schematic block diagram of an example computer network **100** illustratively comprising nodes/devices, such as a plurality of routers/devices interconnected by links or networks, as shown. For example, customer edge (CE) routers **110** may be interconnected with provider edge (PE) routers **120** (e.g., PE-1, PE-2, and PE-3) in order to communicate across a core network, such as an illustrative network backbone **130**. For example, routers **110**, **120** may be interconnected by the public Internet, a multiprotocol label switching (MPLS) virtual private network (VPN), or the like. Data packets **140** (e.g., traffic/messages) may be exchanged among the nodes/devices of the computer network **100** over links using predefined network communication protocols such as the Transmission Control Protocol/Internet Protocol (TCP/IP), User Datagram Protocol (UDP), Asynchronous Transfer Mode (ATM) protocol, Frame Relay protocol, or any other suitable protocol. Those skilled in the art will understand that any number of nodes, devices, links, etc. may be used in the computer network, and that the view shown herein is for simplicity.

In some implementations, a router or a set of routers may be connected to a private network (e.g., dedicated leased lines, an optical network, etc.) or a virtual private network (VPN), such as an MPLS VPN utilizing a Service Provider network, via one or more links exhibiting very different network and service level agreement characteristics. For the sake of illustration, a given customer site may fall under any of the following categories:

1.) Site Type A: a site connected to the network (e.g., via a private or VPN link) using a single CE router and a single link, with potentially a backup link (e.g., a 3G/4G/5G/LTE backup connection). For example, a particular CE router **110** shown in network **100** may support a given customer site, potentially also with a backup link, such as a wireless connection.

2.) Site Type B: a site connected to the network using two MPLS VPN links (e.g., from different Service Providers) using a single CE router, with potentially a backup link (e.g., a 3G/4G/5G/LTE connection). A site of type B may itself be of different types:

2a.) Site Type B1: a site connected to the network using two MPLS VPN links (e.g., from different Service Providers), with potentially a backup link (e.g., a 3G/4G/5G/LTE connection).

2b.) Site Type B2: a site connected to the network using one MPLS VPN link and one link connected to the public Internet, with potentially a backup link (e.g., a 3G/4G/5G/LTE connection). For example, a particular customer site may be connected to network **100** via PE-3 and via a separate Internet connection, potentially also with a wireless backup link.

2c.) Site Type B3: a site connected to the network using two links connected to the public Internet, with potentially a backup link (e.g., a 3G/4G/5G/LTE connection).

Notably, MPLS VPN links are usually tied to a committed service level agreement, whereas Internet links may either have no service level agreement or a loose service level agreement (e.g., a "Gold Package" Internet service connection that guarantees a certain level of performance to a customer site).

3.) Site Type C: a site of type B (e.g., types B1, B2 or B3) but with more than one CE router (e.g., a first CE router connected to one link while a second CE router is connected to the other link), and potentially a backup link (e.g., a wireless 3G/4G/5G/LTE backup link). For example, a particular customer site may include a first CE router **110** connected to PE-2 and a second CE router **110** connected to PE-3.

FIG. **1**B illustrates an example of network **100** in greater detail, according to various embodiments. As shown, network backbone **130** may provide connectivity between devices located in different geographical areas and/or different types of local networks. For example, network **100** may comprise local/branch networks **160**, **162** that include devices/nodes **10-16** and devices/nodes **18-20**, respectively, as well as a data center/cloud environment **150** that includes servers **152-154**. Notably, local networks **160-162** and data center/cloud environment **150** may be located in different geographic locations.

Servers **152-154** may include, in various embodiments, a network management server (NMS), a dynamic host configuration protocol (DHCP) server, a constrained application protocol (CoAP) server, an outage management system (OMS), an application policy infrastructure controller (APIC), an application server, etc. As would be appreciated, network **100** may include any number of local networks, data centers, cloud environments, devices/nodes, servers, etc.

In some embodiments, the techniques herein may be applied to other network topologies and configurations. For example, the techniques herein may be applied to peering points with high-speed links, data centers, etc.

In various embodiments, network **100** may include one or more mesh networks, such as an Internet of Things network. Loosely, the term "Internet of Things" or "IoT" refers to uniquely identifiable objects (things) and their virtual representations in a network-based architecture. In particular, the next frontier in the evolution of the Internet is the ability to connect more than just computers and communications devices, but rather the ability to connect "objects" in general, such as lights, appliances, vehicles, heating, ventilating, and air-conditioning (HVAC), windows and window shades and blinds, doors, locks, etc. The "Internet of Things" thus generally refers to the interconnection of objects (e.g., smart objects), such as sensors and actuators, over a computer network (e.g., via IP), which may be the public Internet or a private network.

Notably, shared-media mesh networks, such as wireless or PLC networks, etc., are often deployed on what are referred to as Low-Power and Lossy Networks (LLNs), which are a class of network in which both the routers and their interconnect are constrained: LLN routers typically operate with constraints, e.g., processing power, memory, and/or energy (battery), and their interconnects are characterized by, illustratively, high loss rates, low data rates, and/or instability. LLNs are comprised of anything from a few dozen to thousands or even millions of LLN routers, and support point-to-point traffic (between devices inside the LLN), point-to-multipoint traffic (from a central control point such at the root node to a subset of devices inside the LLN), and multipoint-to-point traffic (from devices inside the LLN towards a central control point). Often, an IoT network is implemented with an LLN-like architecture. For example, as shown, local network **160** may be an LLN in which CE-2 operates as a root node for nodes/devices **10-16** in the local mesh, in some embodiments.

In contrast to traditional networks, LLNs face a number of communication challenges. First, LLNs communicate over a physical medium that is strongly affected by environmental conditions that change over time. Some examples include temporal changes in interference (e.g., other wireless networks or electrical appliances), physical obstructions (e.g., doors opening/closing, seasonal changes such as the foliage density of trees, etc.), and propagation characteristics of the physical media (e.g., temperature or humidity changes, etc.). The time scales of such temporal changes can range between milliseconds (e.g., transmissions from other transceivers) to months (e.g., seasonal changes of an outdoor environment). In addition, LLN devices typically use low-cost and low-power designs that limit the capabilities of their transceivers.

In particular, LLN transceivers typically provide low throughput. Furthermore, LLN transceivers typically support limited link margin, making the effects of interference and environmental changes visible to link and network protocols. The high number of nodes in LLNs in comparison to traditional networks also makes routing, quality of service (QoS), security, network management, and traffic engineering extremely challenging, to mention a few.

FIG. **2** is a schematic block diagram of an example node/device **200** that may be used with one or more embodiments described herein, e.g., as any of the computing devices shown in FIGS. **1A-1B**, particularly the PE routers **120**, CE routers **110**, nodes/device **10-20**, servers **152-154** (e.g., a network controller located in a data center, etc.), any other computing device that supports the operations of network **100** (e.g., switches, etc.), or any of the other devices referenced below. The device **200** may also be any other suitable type of device depending upon the type of network architecture in place, such as IoT nodes, etc. Device **200** comprises one or more network interfaces **210**, one or more processors **220**, and a memory **240** interconnected by a system bus **250**, and is powered by a power supply **260**.

The network interfaces **210** include the mechanical, electrical, and signaling circuitry for communicating data over physical links coupled to the network **100**. The network interfaces may be configured to transmit and/or receive data using a variety of different communication protocols. Notably, a physical network interface **210** may also be used to implement one or more virtual network interfaces, such as for virtual private network (VPN) access, known to those skilled in the art.

The memory **240** comprises a plurality of storage locations that are addressable by the processor(s) **220** and the network interfaces **210** for storing software programs and data structures associated with the embodiments described herein. The processor **220** may comprise necessary elements or logic adapted to execute the software programs and manipulate the data structures **245**. An operating system **242** (e.g., the Internetworking Operating System, or IOS®, of Cisco Systems, Inc., another operating system, etc.), portions of which are typically resident in memory **240** and executed by the processor(s), functionally organizes the node by, inter alia, invoking network operations in support of software processors and/or services executing on the device. These software processors and/or services may comprise a deep fusion reasoning engine (DFRE) process **248**, as described herein.

It will be apparent to those skilled in the art that other processor and memory types, including various computer-readable media, may be used to store and execute program instructions pertaining to the techniques described herein. Also, while the description illustrates various processes, it is expressly contemplated that various processes may be embodied as modules configured to operate in accordance with the techniques herein (e.g., according to the functionality of a similar process). Further, while processes may be shown and/or described separately, those skilled in the art will appreciate that processes may be routines or modules within other processes.

DFRE process **248** includes computer executable instructions that, when executed by processor(s) **220**, cause device **200** to provide cognitive reasoning services to a network. In various embodiments, DFRE process **248** may utilize machine learning techniques, in whole or in part, to perform its analysis and reasoning functions. In general, machine learning is concerned with the design and the development of techniques that take as input empirical data (such as network statistics and performance indicators) and recognize complex patterns in these data. One very common pattern among machine learning techniques is the use of an underlying model M, whose hyper-parameters are optimized for minimizing the cost function associated to M, given the input data. The learning process then operates by adjusting the hyper-parameters such that the number of misclassified points is minimal. After this optimization phase (or learning phase), the model M can be used very easily to classify new data points. Often, M is a statistical model, and the minimization of the cost function is equivalent to the maximization of the likelihood function, given the input data.

In various embodiments, DFRE process **248** may employ one or more supervised, unsupervised, or self-supervised machine learning models. Generally, supervised learning entails the use of a training large set of data, as noted above, that is used to train the model to apply labels to the input data. For example, in the case of video recognition and analysis, the training data may include sample video data that depicts a certain object and is labeled as such. On the other end of the spectrum are unsupervised techniques that do not require a training set of labels. Notably, while a supervised learning model may look for previously seen patterns that have been labeled as such, an unsupervised model may instead look to whether there are sudden changes in the behavior. Self-supervised is a representation learning approach that eliminates the pre-requisite requiring humans to label data. Self-supervised learning systems extract and use the naturally available relevant context and embedded metadata as supervisory signals. Self-supervised learning models take a middle ground approach: it is different from unsupervised learning as systems do not learn the inherent structure of data, and it is different from supervised learning as systems learn entirely without using explicitly-provided labels.

Example machine learning techniques that DFRE process **248** can employ may include, but are not limited to, nearest neighbor (NN) techniques (e.g., k-NN models, replicator NN models, etc.), statistical techniques (e.g., Bayesian networks, etc.), clustering techniques (e.g., k-means, mean-shift, etc.), neural networks (e.g., reservoir networks, artificial neural networks, etc.), support vector machines (SVMs), logistic or other regression, Markov models or chains, principal component analysis (PCA) (e.g., for linear models), multi-layer perceptron (MLP) artificial neural networks (ANNs) (e.g., for non-linear models), replicating reservoir networks (e.g., for non-linear models, typically for time series), random forest classification, or the like. Accordingly, DFRE process **248** may employ deep learning, in some embodiments. Generally, deep learning is a subset of machine learning that employs ANNs with multiple layers, with a given layer extracting features or transforming the outputs of the prior layer.

The performance of a machine learning model can be evaluated in a number of ways based on the number of true positives, false positives, true negatives, and/or false negatives of the model. For example, the false positives of the model may refer to the number of times the model incorrectly identified an object or condition within a video feed. Conversely, the false negatives of the model may refer to the number of times the model failed to identify an object or condition within a video feed. True negatives and positives may refer to the number of times the model correctly determined that the object or condition was absent in the video or was present in the video, respectively. Related to these measurements are the concepts of recall and precision. Generally, recall refers to the ratio of true positives to the sum of true positives and false negatives, which quantifies the sensitivity of the model. Similarly, precision refers to the ratio of true positives the sum of true and false positives.

According to various embodiments, FIG. **3** illustrates an example hierarchy **300** for a deep fusion reasoning engine (DFRE). For example, DFRE process **248** shown in FIG. **2** may execute a DFRE for any number of purposes. In particular, DFRE process **248** may be configured to analyze sensor data in an IoT deployment (e.g., video data, etc.), to analyze networking data for purposes of network assurance, control, enforcing security policies and detecting threats, facilitating collaboration, or, as described in greater detail below, to aid in the development of a collaborative knowledge generation and learning system for visual programming.

In general, a reasoning engine, also known as a 'semantic reasoner,' 'reasoner,' or 'rules engine,' is a specialized form of machine learning software that uses asserted facts or axioms to infer consequences, logically. Typically, a reasoning engine is a form of inference engine that applies inference rules defined via an ontology language. As introduced herein, a DFRE is an enhanced form of reasoning engine that further leverages the power of sub-symbolic machine learning techniques, such as neural networks (e.g., deep learning), allowing the system to operate across the full spectrum of sub-symbolic data all the way to the symbolic level.

At the lowest layer of hierarchy **300** is sub-symbolic layer **302** that processes the sensor data **312** collected from the network. For example, sensor data **312** may include video feed/stream data from any number of cameras located throughout a location. In some embodiments, sensor data **312** may comprise multimodal sensor data from any number of different types of sensors located throughout the location. At the core of sub-symbolic layer **302** may be one or more DNNs **308** or other machine learning-based model that processes the collected sensor data **312**. In other words, sub-symbolic layer **302** may perform sensor fusion on sensor data **312** to identify hidden relationships between the data.

At the opposing end of hierarchy **300** may be symbolic layer **306** that may leverage symbolic learning. In general, symbolic learning includes a set of symbolic grammar rules specifying the representation language of the system, a set of symbolic inference rules specifying the reasoning competence of the system, and a semantic theory containing the definitions of "meaning." This approach differs from other learning approaches that try to establish generalizations from facts as it is about reasoning and extracting knowledge from knowledge. It combines knowledge representations and reasoning to acquire and ground knowledge from observations in a non-axiomatic way. In other words, in sharp contrast to the sub-symbolic learning performed in layer **302**, the symbolic learning and generalized intelligence performed at symbolic layer **306** requires a variety of reasoning and learning paradigms that more closely follows how humans learn and are able to explain why a particular conclusion was reached.

Symbolic learning models what are referred to as "concepts," which comprise a set of properties. Typically, these properties include an "intent" and an "extent," whereby the intent offers a symbolic way of identifying the extent of the concept. For example, consider the intent that represents motorcycles. The intent for this concept may be defined by properties such as "having two wheels" and "motorized," which can be used to identify the extent of the concept (e.g., whether a particular vehicle is a motorcycle).

Linking sub-symbolic layer **302** and symbolic layer **306** may be conceptual layer **304** that leverages conceptual spaces. In general, conceptual spaces are a proposed framework for knowledge representation by a cognitive system on the conceptual level that provides a natural way of representing similarities. Conceptual spaces enable the interaction between different type of data representations as an intermediate level between sub-symbolic and symbolic representations.

More formally, a conceptual space is a geometrical structure which is defined by a set of quality dimensions to allow for the measurement of semantic distances between instances of concepts and for the assignment of quality values to their quality dimensions, which correspond to the properties of the concepts. Thus, a point in a conceptual space S may be represented by an n-dimensional conceptual vector $v=<d_1, \ldots, d_i, \ldots, d_n>$ where d_i represents the quality value for the i^{th} quality dimension. For example, consider the concept of taste. A conceptual space for taste may include the following dimensions: sweet, sour, bitter, and salty, each of which may be its own dimension in the conceptual space. The taste of a given food can then be represented as a vector of these qualities in a given space (e.g., ice cream may fall farther along the sweet dimension than that of peanut butter, peanut butter may fall farther along the salty dimension than that of ice cream, etc.). By representing concepts within a geometric conceptual space, similarities can be compared in geometric terms, based on the Manhattan distance between domains or the Euclidean distance within a domain in the space. In addition, similar objects can be grouped into meaningful conceptual space regions through the application of clustering techniques, which extract concepts from data (e.g., observations).

Said differently, a conceptual space is a framework for representing information that models human-like reasoning to compose concepts using other existing concepts. Note that these representations are not competing with symbolic or associationistic representations. Rather, the three kinds can be seen as three levels of representations of cognition with different scales of resolution and complementary. Namely, a conceptual space is built up from geometrical representations based on a number of quality dimensions that complements the symbolic and deep learning models of symbolic layer **306** and sub-symbolic layer **302**, representing an operational bridge between them. Each quality dimension may also include any number of attributes, which present other features of objects in a metric subspace based on their measured quality values. Here, similarity between concepts is just a matter of metric distance between them in the conceptual space in which they are embedded.

In other words, a conceptual space is a geometrical representation which allows the discovery of regions that are physically or functionally linked to each other and to abstract symbols used in symbolic layer **306**, allowing for the discovery of correlations shared by the conceptual domains during concepts formation. For example, an alert prioritization module may use connectivity to directly acquire and evaluate alerts as evidence. Possible enhancements may include using volume of alerts and novelty of adjacent (spatially/temporally) alerts, to tune level of alertness.

In general, the conceptual space at conceptual layer **304** allows for the discovery of regions that are naturally linked to abstract symbols used in symbolic layer **306**. The overall model is bi-directional as it is planned for predictions and action prescriptions depending on the data causing the activation in sub-symbolic layer **302**.

Layer hierarchy **300** shown is particularly appealing when matched with the attention mechanism provided by a cognitive system that operates under the assumption of limited resources and time-constraints. For practical applications, the reasoning logic in symbolic layer **306** may be non-axiomatic and constructed around the assumption of insufficient knowledge and resources (AIKR). It may be implemented, for example, with a Non-Axiomatic Reasoning System (open-NARS) **310**. However, other reasoning engines can also be used, such as Auto-catalytic Endogenous Reflective Architecture (AERA), OpenCog, and the like, in symbolic layer **306**, in further embodiments. Even Prolog may be suitable, in some cases, to implement a reasoning engine in symbolic layer **306**. In turn, an output **314** coming from symbolic layer **306** may be provided to a user interface (UI) for review. For example, output **314** may comprise a video feed/stream augmented with inferences or conclusions made by the DFRE, such as the locations of unstocked or under-stocked shelves, etc.

By way of example of symbolic reasoning, consider the ancient Greek syllogism: (1.) All men are mortal, (2.) Socrates is a man, and (3.) therefore, Socrates is mortal. Depending on the formal language used for the symbolic reasoner, these statements can be represented as symbols of a term logic. For example, the first statement can be represented as "man→[mortal]" and the second statement can be represented as "{Socrates}→man." Thus, the relationship between terms can be used by the reasoner to make inferences and arrive at a conclusion (e.g., "Socrates is mortal"). Non-axiomatic reasoning systems (NARS) generally differ from more traditional axiomatic reasoners in that the former applies a truth value to each statement, based on the amount of evidence available and observations retrieved, while the latter relies on axioms that are treated as a baseline of truth from which inferences and conclusions can be made.

In other words, a DFRE generally refers to a cognitive engine capable of taking sub-symbolic data as input (e.g., raw or processed sensor data regarding a monitored system), recognizing symbolic concepts from that data, and applying symbolic reasoning to the concepts, to draw conclusions about the monitored system.

According to various embodiments, FIG. **4** illustrates an example DFRE architecture **400**. As shown, architecture **400** may be implemented across any number of devices or fully on a particular device, as desired. At the core of architecture **400** may be DFRE middleware **402** that offers a collection of services, each of which may have its own interface. In general, DFRE middleware **402** may leverage a library for interfacing, configuring, and orchestrating each service of DFRE middleware **402**.

In various embodiments, DFRE middleware **402** may also provide services to support semantic reasoning, such as by an AIKR reasoner. For example, as shown, DFRE middleware **402** may include a NARS agent that performs semantic reasoning for structural learning. In other embodiments, OpenCog or another suitable AIKR semantic reasoner could be used.

One or more DFRE agents **404** may interface with DFRE middleware **402** to orchestrate the various services available from DFRE middleware **402**. In addition, DFRE agent **404** may feed and interact with the AIKR reasoner so as to populate and leverage a DFRE knowledge graph with knowledge.

More specifically, in various embodiments, DFRE middleware **402** may obtain sub-symbolic data **408**. In turn, DFRE middleware **402** may leverage various ontologies, programs, rules, and/or structured text **410** to translate sub-symbolic data **408** into symbolic data **412** for consumption by DFRE agent **404**. This allows DFRE agent **404** to apply symbolic reasoning to symbolic data **412**, to populate and update a DFRE knowledge base (KB) **416** with knowledge **414** regarding the problem space (e.g., the network under observation, etc.). In addition, DFRE agent **404** can leverage the stored knowledge **414** in DFRE KB **416** to make assessments/inferences.

For example, DFRE agent **404** may perform semantic graph decomposition on DFRE KB **416** (e.g., a knowledge graph), so as to compute a graph from the knowledge graph of KB **416** that addresses a particular problem. DFRE agent **404** may also perform post-processing on DFRE KB **416**, such as performing graph cleanup, applying deterministic rules and logic to the graph, and the like. DFRE agent **404** may further employ a definition of done, to check goals and collect answers using DFRE KB **416**.

In general, DFRE KB **416** may comprise any or all of the following:
- Data
- Ontologies
- Evolutionary steps of reasoning
- Knowledge (e.g., in the form of a knowledge graph)
- The Knowledge graph also allows different reasoners to:
  - Have their internal subgraphs
  - Share or coalesce knowledge
  - Work cooperatively

In other words, DFRE KB **416** acts as a dynamic and generic memory structure. In some embodiments, DFRE KB **416** may also allow different reasoners to share or coalesce knowledge, have their own internal sub-graphs, and/or work collaboratively in a distributed manner. For example, a first DFRE agent **404** may perform reasoning on a first sub-graph, a second DFRE agent **404** may perform reasoning on a second sub-graph, etc., to evaluate the health of the network and/or find solutions to any detected problems. To communicate with DFRE agent **404**, DFRE KB **416** may include a bidirectional Narsese interface or other interface using another suitable grammar.

In various embodiments, DFRE KB **416** can be visualized on a user interface. For example, Cytoscape, which has its building blocks in bioinformatics and genomics, can be used to implement graph analytics and visualizations.

Said differently, DFRE architecture **400** may include any or all of the following components:
- DFRE middleware **402** that comprises:
  - Structural learning component
  - JSON, textual data, ML/DL pipelines, and/or other containerized services (e.g., using Docker)
  - Hierarchical goal support
- DFRE Knowledge Base (KB) **416** that supports:
  - Bidirectional Narseseese interface
  - Semantic graph decomposition algorithms
  - Graph analytics
  - Visualization services
- DFRE Agent **404**
  - DFRE Control System

More specifically, in some embodiments, DFRE middleware **402** may include any or all of the following:
- Subsymbolic services:
  - Data services to collect sub-symbolic data for consumption
- Reasoner(s) for structural learning
- NARS
- OpenCog
- Optimized hierarchical goal execution
  - Probabilistic programming

Causal inference engines

Visualization Services (e.g., Cytoscape, etc.)

DFRE middleware **402** may also allow the addition of new services needed by different problem domains.

During execution, DFRE agent **404** may, thus, perform any or all of the following:

- Orchestration of services
- Focus of attention
  - Semantic graph decomposition
    - Addresses combinatorial issues via an automated divide and conquer approach that works even in non-separable problems because the overall knowledge graph **416** may allow for overlap.
- Feeding and interacting with the AIKR reasoner via bidirectional translation layer to the DFRE knowledge graph.
  - Call middleware services
- Post processing of the graph
  - Graph clean-up
  - Apply deterministic rules and logic to the graph
- Definition of Done (DoD)
  - Check goals and collect answers

FIG. **5** illustrates an example **500** showing the different forms of structural learning that the DFRE framework can employ. More specifically, the inference rules in example **500** relate premises S→M and M→P, leading to a conclusion SP. Using these rules, the structural learning herein can be implemented using an ontology with respect to an Assumption of Insufficient Knowledge and Resources (AIKR) reasoning engine, as noted previously. This allows the system to rely on finite processing capacity in real time and be prepared for unexpected tasks. More specifically, as shown, the DFRE may support any or all of the following:

- Syllogistic Logic
  - Logical quantifiers
- Various Reasoning Types
  - Deduction Induction
  - Abduction
  - Induction
  - Revision
- Different Types of Inference
- Local inference
- Backward inference

To address combinatorial explosion, the DFRE knowledge graph may be partitioned such that each partition is processed by one or more DFRE agents **404**, as shown in FIG. **6**, in some embodiments. More specifically, any number of DFRE agents **404** (e.g., a first DFRE agent **404***a* through an N^{th} DFRE agent **404***n*) may be executed by devices connected via a network **602** or by the same device. In some embodiments, DFRE agents **404***a*-**404***n* may be deployed to different platforms (e.g., platforms **604***a*-**604***n*) and/or utilize different learning approaches. For instance, DFRE agent **404***a* may leverage neural networks **606**, DFRE agent **404***b* may leverage Bayesian learning **608**, DFRE agent **404***c* may leverage statistical learning, and DFRE agent **404***n* may leverage decision tree learning **612**.

As would be appreciated, graph decomposition can be based on any or all of the following:

- Spatial relations—for instance, this could include the vertical industry of a customer, physical location (country) of a network, scale of a network deployment, or the like.
- Descriptive properties, such as severity, service impact, next step, etc.
- Graph-based components (isolated subgraphs, minimum spanning trees, all shortest paths, strongly connected components . . .)

Any new knowledge and related reasoning steps can also be input back to the knowledge graph, in various embodiments.

In further embodiments, the DFRE framework may also support various user interface functions, so as to provide visualizations, actions, etc. to the user. To do so, the framework may leverage Cytoscape, web services, or any other suitable mechanism.

At the core of the techniques herein is a knowledge representation metamodel **700** for different levels of abstraction, as shown in FIG. **7**, according to various embodiments. In various embodiments, the DFRE knowledge graph groups information into four different levels, which are labeled L_0, L_1, L_2, and L* and represent different levels of abstraction, with L_0 being closest to raw data coming in from various sensors and external systems and L_2 representing the highest levels of abstraction typically obtained via mathematical means such as statistical learning and reasoning. L* can be viewed as the layer where high-level goals and motivations are stored. The overall structure of this knowledge is also based on anti-symmetric and symmetric relations.

One key advantage of the DFRE knowledge graph is that human level domain expertise, ontologies, and goals are entered at the L_2 level. This leads, by definition, to an unprecedented ability to generalize at the L_2 level thus minimizing the manual effort required to ingest domain expertise.

More formally:

- L* represents the overall status of the abstraction. In case of a problem, it triggers problem solving in lower layers via a DFRE agent **702**.
- $L_{2.1}$-$L_{2.\infty}$=Higher level representations of the world in which most of concepts and relations are collapsed into simpler representations. The higher-level representations are domain-specific representations of lower levels.
- L_1=has descriptive, teleological and structural information about L_0.
- L_0=Object level is the symbolic representation of the physical world.

In various embodiments, L_2 may comprise both expertise and experience stored in long-term memory, as well as a focus of attention (FOA) in short-term memory. In other words, when a problem is triggered at L*, a DFRE agent **702** that operates on L_2-L_0 may control the FOA so as to focus on different things, in some embodiments.

As would be appreciated, there may be hundreds of thousands or even millions of data points that need to be extracted at L_0. The DFRE's FOA is based on the abstraction and the DFRE knowledge graph (KG) may be used to keep combinatorial explosion under control.

Said differently, metamodel **700** may generally take the form of a knowledge graph in which semantic knowledge is stored regarding a particular system, such as a computer network and its constituent networking devices. By representing the relationships between such real-world entities (e.g., router A, router B, etc.), as well as their more abstract concepts (e.g., a networking router), DFRE agent **702** can make evaluations regarding the particular system at different levels of extraction. Indeed, metamodel **700** may differ from a more traditional knowledge graph through the inclusion of any or all of the following, in various embodiments:

A formal mechanism to represent different levels of abstraction, and for moving up and down the abstraction hierarchy (e.g., ranging from extension to intension).

Additional structure that leverages distinctions/anti-symmetric relations, as the backbone of the knowledge structures.

Similarity/symmetric relation-based relations.

As noted above, the creation of models, ontologies, diagrams, software programs, and other similar artifacts remains a very time consuming and resource intensive activity. While there have been many attempts at creating visual programming systems, these have largely failed due to the relatively low productivity that results from their point-and-click interfaces. Another source of failure of these visual programming systems has been caused by the referential integrity problem. More specifically, once an artifact is created via the visual programming system, any editing of the artifact cannot be automatically parsed into the visual representation.

Although new computer languages with higher expressive power are continually being introduced, e.g., functional programming languages such as Haskell and Rust, these more expressive languages also require increasing sophistication on the part of the software developer. This limits the usability of these more expressive languages. In addition, the gap between the supply of computer scientists, programmers, and other experts that can produce high quality digital artifacts and the demand for such individuals is growing rapidly. Further, the extreme programming and pair programming movements, along with recent moves towards more remote working, has also increased the demand for improved mechanisms for collaborating and joint development of these work products.

—Collaborative Visual Programming Environment with Cumulative Learning Using a DFRE—

The techniques herein introduce a deep fusion reasoning engine (DFRE)-based system to support a multimodal visual programming environment. In some aspects, the techniques herein allow for increased levels of productivity and collaboration in a visual programming environment with respect to the creation and modification of digital artifacts within the environment (e.g., models, ontologies, diagrams, documents, software, etc.).

Illustratively, the techniques described herein may be performed by hardware, software, and/or firmware, such as in accordance with the DFRE process **248**, which may include computer executable instructions executed by the processor **220** (or independent processor of interfaces **210**), to perform functions relating to the techniques described herein.

Specifically, according to various embodiments, a device obtains data models and workflow logic for a visual programming environment. The device constructs, based on the data models and workflow logic for the visual programming environment, a metamodel that comprises a knowledge graph. The device makes, using the metamodel, an evaluation of an interaction between a user and the visual programming environment. The device provides, based on the evaluation, visualization data to a user interface of the visual programming environment.

Operationally, FIG. **8** illustrates an example DFRE-based architecture **800** for a visual programming environment, according to various embodiments. As would be appreciated, visual programming allows a user to graphically define and edit a program through the manipulation of a graphical display (e.g., via a point-and-click interface, etc.). This is in contrast to more traditional approaches that are text-based and require the user to know and understand the syntax, operators, etc. of the programming language. While some text-based programming languages have implemented some graphical manipulations into their environments (e.g., Visual Basic, Visual C++, etc.), these languages are still text-based at their core. In contrast, visual programming environments typically rely on the manipulation of graphical symbols, to specify the control flow of the program. For instance, this is typically done through the specification of a dataflow diagram in the visual programming environment (e.g., a flow diagram linking with directed arrows between steps/functions, etc.).

At the core of architecture **800** may be a DFRE multimodal visual programming agent **802** that leverages a metamodel, as described previously, to make semantic inferences about a visual programming environment. In turn, agent **802** may provide visualization data for presentation to a user via a user interface of the visual programming environment. Indeed, by learning the data models, workflow logic, etc. associated with the visual programming environment, agent **802** can aid the user in creating or editing a program via the visual programming environment.

More specifically, as shown, assume that a programmer **806** interacts with a point & click user interface (UI) **804** of the visual programming environment supported by visual programming services **812**. For instance, programmer **806** may begin creating a new dataflow diagram in the visual programming environment via UI **804**. In turn, visual programming services **812** may provide data regarding this to agent **802**, which can extract knowledge regarding the interactions (e.g., by parsing the visual program) and evaluating the resulting knowledge representation **814** using its knowledge base (KB) **808** of workflows and software stored in its metamodel. For instance, agent **802** may identify the intent of programmer **806** through their specification of one or more dataflow diagram blocks, connectors, etc. via UI **804**. In turn, agent **802** may provide additional visualization data to UI **804**, such as, but not limited to, suggested changes to the visual program, contextual data that can aid programmer **806** (e.g., an instruction manual or other documentation), data from the system to be controlled by the visual program, or the like.

In some embodiments, UI **804** may take the form of one or more electronic displays on which the visual programming environment is presented to programmer **806**. However, in further embodiments, UI **804** may also take the form of a more advance UI, such as an augmented reality display, mixed reality display, virtual reality display, or other "XR" display.

In an additional embodiment, programmer **806** may also interact with the visual programming environment via speech to text conversion services **818**, allowing programmer **806** to issue audio command and queries to the visual programming environment. In such a case, DFRE-based architecture **800** may also include Natural Language Processing (NLP) services **816** that parse any captured audio and provide information regarding the audio interactions to agent **802**. Similar to its evaluations of any interactions of programmer **806** with UI **804**, agent **802** may also evaluate any audio interactions and present data to programmer **806**, accordingly (e.g., by sending visualization data to UI **804**, as speech via a text-to-speech converter, etc.).

By way of example, assume that the visual programming environment is used to program/configure networking devices in a computer network (e.g., routers, switches, etc.). In this case, programmer **806** may begin by saying "bring up the network topology diagram for customer X at site Y." In response, agent **802** may search its KB **808** for the requested topology information and present it to the user via UI **804**. Note that if the requested information is not in KB **808**, agent **802** could also retrieve the requested information from an external source, such as a network controller for the network at site Y, if so configured (e.g., KB **808** does not directly store the topology, but instead stores knowledge about the concepts of a network topology diagram, the concept of a network controller or other source of this information from which to obtain the information, etc.). Preliminary testing of the techniques herein has shown that the capability of agent **802** to use its metamodel/KB **808** to recognize and assess the user context, customer context, natural language, along with prior artifacts entered into the visual programming environment, provides far greater accuracy than other approaches that rely solely on machine learning or deep learning.

Once a target work artifact is loaded into the DFRE metamodel/KB **808**, agent **802** may also generate an editable, visual representations or diagram of that artifact for presentation via UI **804**. For instance, in the case of a network topology, agent **802** may convert the network topology into a set of blocks and connectors that represent the various functions of the networking devices and their relationships for presentation in UI **804**. This allows programmer **806** to modify the visual representation via UI **804**, such as to reconfigure the network by manipulating the visual representation in the visual programming environment.

Agent **802** may update the DFRE metamodel/KB **808** in real time, based on any changes made by programmer **806** within the visual programming environment. As would be appreciated, KB **808** may be initially seeded through an LTM interface **810** for ontologies and expert knowledge via which a domain expert may craft the initial knowledge base for agent **802**, as detailed below.

Also as shown, DFRE-based architecture **800** may include DFRE structured text services **820** that are configured to convert the visual program into its corresponding commands, configurations, etc. for deployment to live environment **824** (e.g., a computer network, etc.). In some embodiments, prior to fully deploying them to live environment **824**, they may first be sent by services **820** to a testbed environment **822**. This allows the system to validate the commands, configurations, etc. and provide feedback to UI **804** via agent **802**. For instance, if the resulting changes cause an error in testbed environment **822**, agent **802** may assess the error using its metamodel/KB **808** and make recommendations to programmer **806**, accordingly.

FIG. **9** illustrates an example **900** of the interactions of a programmer **902** with a visual programming environment, according to various embodiments. As shown, programmer **902** may interact with a point & click UI **904** of the visual programming environment and, potentially, via an NLP conversational interface **910**. In turn, the visual programming environment may provide visualization data **906** for presentation to programmer **902**, such as a dataflow diagram.

By analyzing the diagram semantics **906***a* from visualization data **906** using its metamodel/KB, DFRE agent **908** may provide diagram suggestions **906***b*, such as a change **912**. This allows programmer **902** to review the proposed change and either accept it, thereby replacing the current diagram or rejecting it.

FIG. **10** illustrates an example data pipeline **1000** to populate a DFRE knowledge base, such as KB **808** described previously, according to various embodiments. As shown, data pipeline **1000** may comprise an ontology editor **1004** that is responsible for generating ontologies **1018** that are published to KB **808**. For instance, ontology editor **1004** may be used to implement interface **810** described previously in FIG. **8**.

During use, ontology editor **1004** may receive workflow logic **1002** via a graphical user interface (GUI **1006**). Typically, one or more domain experts, such as a network engineer, a technical assistance center (TAC) expert, or the like, may provide workflow logic **1002** to ontology editor **1004**, which is used as input to compiler **1008**. In general, workflow logic **1002** may specify the various workflows of the system to be controlled using the visual programming environment. For instance, the domain expert may specify the steps and/or dataflow diagram to troubleshoot and correct an error in the network as workflow logic **1002**. In further embodiments, workflow logic **1002** may also include any of the artifacts created by the domain expert within the visual programming environment.

Also as shown, ontology editor **1004** may also receive data models **1020** regarding the system to be controlled using the visual programming environment. For instance, in the case of a computer network, data models **1020** can be obtained from the software development kit (SDK) for the networking devices of the network, any data modeling language files associated with the network, such as Yet Another Next Generation (YANG) models, and the like.

According to various embodiments, ontology editor **1004** may include a semantic mapper **1010** that maps workflow logic **1002** and ontology editor **1004** into semantic knowledge, such as concepts **1014** and their relationships **1016**, for inclusion in the ontologies **1018** that are published to KB **808**. In addition, from the specific workflow logic **1002**, rules **1012** can also be codified by ontology editor **1004** for inclusion in ontologies **1018**. By way of example, assume that data models **1020** includes information regarding the various calls, functions, etc. needed to communicate control command to a router. In such a case, workflow logic **1002** may specify which of those calls, functions, etc. should be used and under which circumstances. In turn, ontology editor **1004** may codify this information as rules **1012** and used to seed ontologies **1018** for KB **808**.

FIG. **11** illustrates an example **1100** of the collaboration of users in a visual programming environment, according to various embodiments. As noted previously with respect to FIG. **6**, the metamodel and DFRE architecture used herein is also able to support cumulative and distributed learning across any number of devices and users. For instance, assume that users **1104***a*-**1104***d* each interact with visual programming environment **1106** using their own devices. As shown, each of these devices may execute its own corresponding DFRE agents **1102***a*-**1102***d*.

Using the techniques above, DFRE agents **1102***a*-**1102***d* may share knowledge with one another, thereby allowing the system to learn from the interactions with each of users **1104***a*-**1104***d*. For instance, assume that user **1104***a* is an expert user that has created a certain dataflow diagram within visual programming environment **1106**. Accordingly, DFRE agent **1102***a* may acquire knowledge about this artifact and share its knowledge about the artifact with the other DFRE agents **1102***b*-**1102***d*. Such updating can be performed synchronously or asynchronously, as desired. Thus, if another user, such as user **1104***c*, attempts to create a similar dataflow diagram, DFRE agent **1102***c* may provide suggestions for review by user **1104***c*, effectively sharing the knowledge and expertise of user **1104***a* with user **1104***c*.

FIG. **12** illustrates an example simplified procedure for using a DFRE with a visual programming environment, in accordance with one or more embodiments described herein. For example, a non-generic, specifically configured device (e.g., device **200**) may perform procedure **1200** by executing stored instructions (e.g., DFRE process **248**). The procedure **1200** may start at step **1205**, and continues to step **1210**, where, as described in greater detail above, the device may obtain data models and workflow logic for a visual programming environment. For instance, such a visual programming environment may be used to configure an associated system, such as networking devices in a network, etc. In general, the workflow logic may be specified by one or more domain experts for the associated system. For instance, one workflow may specify the steps that a user may take to troubleshoot and correct problems in a network. Similarly, the data models may

At step **1215**, as detailed above, the device may, based on the data models and workflow logic for the visual programming environment, a metamodel that comprises a knowledge graph. In general, the metamodel comprises a semantic representation of the data models and workflow logic for the visual programming environment. In various embodiments, the metamodel may also include mechanisms to represent the visual programming environment at different levels of abstraction and for moving up and down the abstraction hierarchy, to include symmetric relations between concepts in the knowledge graph, etc. As would be appreciated, the metamodel may also be updated over time, such as through users interacting with the visual programming environment, changes being made to the system being controlled by the visual programming environment, and the like.

At step **1220**, the device may make, using the metamodel, an evaluation of an interaction between a user and the visual programming environment, as described in greater detail above. For instance, the interaction may take the form of a voice command or graphical interaction with the visual programming environment by the user. In response, the device may associate the interaction with a concept in the knowledge graph of the metamodel and use a semantic reasoning engine to make an inference about the concept (e.g., to discern an intent of the user).

At step **1225**, as detailed above, the device may then provide, based on the evaluation, visualization data to a user interface of the visual programming environment. In general, the visualization data may comprise an artifact or a correction to an artifact of the visual programming environment. For instance, in the case of the user making a configuration change to a computer network via the visual programming environment, the visualization data may be a suggested change to the change entered by the user (e.g., based on prior work products, relevant ontologies, etc. in the metamodel). In further embodiments, the visualization data may take the form of additional information to be displayed (or otherwise conveyed, such as audio) to the user via the visual programming environment. For instance, depending on the context of the interaction with the visual programming environment by the user, the device may display an instruction manual or other document, information regarding the system being controlled by the visual programming environment (e.g., current or prior telemetry data, etc.), or the like, to aid the user. Procedure **1200** then ends at step **1230**.

It should be noted that while certain steps within procedure **1200** may be optional as described above, the steps shown in FIG. **12** are merely examples for illustration, and certain other steps may be included or excluded as desired. Further, while a particular order of the steps is shown, this ordering is merely illustrative, and any suitable arrangement of the steps may be utilized without departing from the scope of the embodiments herein.

The techniques herein, therefore, introduce a deep fusion reasoning engine (DFRE) configured to enhance a visual programming environment. In some aspects, the techniques herein may acquire knowledge about the visual programming environment and use the acquired knowledge to aid a user interacting with the visual programming environment. In further aspects, the techniques herein could also be used to implement distributed learning across multiple users, so as to collaboratively capture and leverage knowledge from these users.

While there have been shown and described illustrative embodiments that provide for a DFRE for collaborative knowledge generation and learning, it is to be understood that various other adaptations and modifications may be made within the spirit and scope of the embodiments herein. For example, while certain embodiments are described herein with respect to a visual programming environment, the techniques can be extended without undue experimentation to other programming or configuration environments, as well.

The foregoing description has been directed to specific embodiments. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. For instance, it is expressly contemplated that the components and/or elements described herein can be implemented as software being stored on a tangible (non-transitory) computer-readable medium (e.g., disks/CDs/RAM/EEPROM/etc.) having program instructions executing on a computer, hardware, firmware, or a combination thereof. Accordingly, this description is to be taken only by way of example and not to otherwise limit the scope of the embodiments herein. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the embodiments herein.

What is claimed is:

1. A method comprising:

obtaining, by a device, data models and workflow logic for a visual programming environment, wherein the device obtains the data models and workflow logic from a plurality of distributed users, and the workflow logic comprises one or more workflows for a network to be controlled using the visual programming environment;

constructing, by the device and based on the data models and workflow logic for the visual programming environment, a metamodel that comprises a knowledge graph;

making, by the device and using the metamodel, an evaluation of an interaction between a user and the visual programming environment; and

providing, by the device and based on the evaluation, visualization data to a user interface of the visual programming environment.

2. The method as in claim **1**, wherein the interaction comprises a voice command.

3. The method as in claim **1**, wherein the user interface comprises one of: an augmented reality display, a mixed reality display, or a virtual reality display.

4. The method as in claim **1**, wherein making the evaluation of the interaction between the user and the visual programming environment comprises:

associating the interaction between the user and the visual programming environment with a concept in the knowledge graph; and

using a semantic reasoning engine to make an inference regarding the concept, wherein the visualization data is selected based in part on the inference.

5. The method as in claim **1**, wherein the metamodel comprises a semantic representation of the data models and workflow logic for the visual programming environment.

6. The method as in claim **5**, wherein the metamodel comprises symmetric relations between concepts in the knowledge graph.

7. The method as in claim **5**, wherein the metamodel represents the visual programming environment at different levels of abstraction.

8. The method as in claim **1**, wherein the visualization data indicates a change to a visual program being edited by the user in the visual programming environment.

9. The method as in claim **1**, wherein the visual programming environment is used to configure networking devices in the network.

10. An apparatus, comprising:

a network interface to communicate with a computer network;

a processor coupled to the network interface and configured to execute one or more processes; and

a memory configured to store a process that is executed by the processor, the process when executed configured to:

obtain data models and workflow logic for a visual programming environment, wherein the device obtains the data models and workflow logic from a plurality of distributed users, and the workflow logic comprises one or more workflows for a network to be controlled using the visual programming environment;

construct, based on the data models and workflow logic for the visual programming environment, a metamodel that comprises a knowledge graph;

make, using the metamodel, an evaluation of an interaction between a user and the visual programming environment; and

provide, based on the evaluation, visualization data to a user interface of the visual programming environment.

11. The apparatus as in claim **10**, wherein the interaction comprises a voice command.

12. The apparatus as in claim **10**, wherein the user interface comprises one of: an augmented reality display, a mixed reality display, or a virtual reality display.

13. The apparatus as in claim **10**, wherein the apparatus makes the evaluation of the interaction between the user and the visual programming environment by:

associating the interaction between the user and the visual programming environment with a concept in the knowledge graph; and

using a semantic reasoning engine to make an inference regarding the concept, wherein the visualization data is selected based in part on the inference.

14. The apparatus as in claim **10**, wherein the metamodel comprises a semantic representation of the data models and workflow logic for the visual programming environment.

15. The apparatus as in claim **14**, wherein the metamodel comprises symmetric relations between concepts in the knowledge graph.

16. The apparatus as in claim **14**, wherein the metamodel represents the visual programming environment at different levels of abstraction.

17. The apparatus as in claim **10**, wherein the visualization data indicates a change to a visual program being edited by the user in the visual programming environment.

18. A tangible, non-transitory, computer-readable medium storing program instructions that cause a device to execute a process comprising:

obtaining, by the device, data models and workflow logic for a visual programming environment, wherein the device obtains the data models and workflow logic from a plurality of distributed users, and the workflow logic comprises one or more workflows for a network to be controlled using the visual programming environment;

constructing, by the device and based on the data models and workflow logic for the visual programming environment, a metamodel that comprises a knowledge graph;

making, by the device and using the metamodel, an evaluation of an interaction between a user and the visual programming environment; and

providing, by the device and based on the evaluation, visualization data to a user interface of the visual programming environment.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 11,693,632 B2
APPLICATION NO. : 17/157027
DATED : July 4, 2023
INVENTOR(S) : Hugo Latapie

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In the Specification

Column 11, Line 28, please amend as shown:
sion S→P. Using these rules, the structural learning herein can

Signed and Sealed this
Twenty-sixth Day of September, 2023

Katherine Kelly Vidal

Katherine Kelly Vidal
Director of the United States Patent and Trademark Office